5-1-2008

Anchor-Free Localization in Mixed Wireless Sensor Network Systems

Yurong Xu
Dartmouth College

Dartmouth Computer Science Technical Report TR2008-626

Anchor-Free Localization in Mixed Wireless Sensor Network Systems

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Doctor of Philosophy

in

Computer Science

by

Yurong Xu

DARTMOUTH COLLEGE

Hanover, New Hampshire

May, 2008

Examining Committee:

(chair) Fillia S. Makedon

David Kotz

James Ford

Qun Li

Charles K. Barlowe, Ph.D
Dean of Graduate Studies

Abstract of the Dissertation

Recent technological advances have fostered the emergence of Wireless Sensor Networks (WSNs), which consist of tiny, wireless, battery-powered nodes that are expected to revolutionize the ways in which we understand and construct complex physical systems. A fundamental property needed to use and maintain these WSNs is "localization", which allows the establishment of spatial relationships among nodes over time.

This dissertation presents a series of Geographic Distributed Localization (GDL) algorithms for mixed WSNs, in which both static and mobile nodes can coexist. The GDL algorithms provide a series of useful methods for localization in mixed WSNs. First, GDL provides an approximation called "hop-coordinates", which improves the accuracy of both hop-counting and connectivity-based measurement techniques. Second, GDL utilizes a distributed algorithm to compute the locations of all nodes in static networks with the help of the hop-coordinates approximation. Third, GDL integrates a sensor component into this localization paradigm for possible mobility and as a result allows for a more complex deployment of WSNs as well as lower costs. In addition, the development of GDL incorporated the possibility of manipulated communications, such as wormhole attacks. Simulations show that such a localization system can provide fundamental support for security by detecting and localizing wormhole attacks.

Although several localization techniques have been proposed in the past few years, none currently satisfies our requirements to provide an accurate, efficient and reliable localization for mixed WSNs. The contributions of this dissertation are: (1) our measurement technique achieves better accuracy both in measurement and localization than other methods; (2) our method significantly improves the efficiency of localization in updating location in mixed WSNs by incorporating sensors into the method; (3) our method can detect and locate the communication that has been manipulated by a wormhole in a network without relying on a central server.

Acknowledgments

A large number of people were very important for the completion of this thesis work, and I would like to acknowledge all of them, while I can't include all of their names here.

First, thanks are due to my parents for their life-long support and unconditional love, both of which have been very important to me while doing my studies, and my close friend Hanzhang Pan.

My most endless thanks are due to my advisor, Fillia S. Makedon.

I would like to acknowledge the other members of my thesis committee:

David Kotz, James Ford, Qun Li.

Thanks Prof. David Kotz for his valuable comments and advices. Thanks Prof. James Ford for his continuous help in my thesis writing. Thanks Prof. Qun Li for his creative ideas and strong supports on my thesis.

Also, I would like to thank all the professors at Dartmouth College, and my wonderful colleagues and friends. In particular, I would like to thank:

Former and current members from Dartmouth Experimental Visualization Laboratory: James Ford, Li Shen, Yuhang Wang, Tilmann Steinberg, Heng Huang, Song Ye, Yan Zhao, Zhengyi Le, Yi Ouyang, Sheng Zhang, Wei Zheng, Fei Xiong, Weihong Wang, Zhifeng Wang, Rong Zhang, Vangjel Meci and Eric Becker. And members from Dartmouth Advanced Image Laboratory: Justin Pearlman, Ling Gao, Lynn H. Lee, and MRI technicians in DHMC.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, we give an overview of this dissertation, as well as a motivation of the problem and a summary of the contributions of this dissertation.

1.1 Motivation: The Importance of Localization in WSNs

Context-awareness is an area with a long research history, with the emergence of wearable computing, mobile computing, and ubiquitous/pervasive computing in past decades. Location-awareness is a part of context-awareness that improves these technologies into more efficient, flexible levels, and enables more and more application fields.

With recent advances in Micro-Electro Mechanical Systems (MEMS) technology, wireless communications and integrated circuit fabrication, Wireless Sensor Networks (WSNs) [1, 93] are an emerging new type of ad-hoc networks. WSNs integrate sensing, processing and wireless communication in distributed systems being used in different fields. Through advanced mesh networking protocols, tiny sensor nodes form a new community of connectivity that extends the reach of cyberspace out into the physical world. WSNs have been applied in numerous real-world applications [90], such as surveillance, healthcare, inven-

tory tracking, industry automation, military uses and security. Each node in a WSN has limited capabilities, but when connected as ad hoc distributed system, it is capable of co-operative processing and communication. WSNs integrate the capabilities of small nodes into a large distributed network performing the tasks of not only current applications, but also future applications.

Localization is defined to be the determination of the physical location of each network node in a geographic map, or finding its relative location from a topology map in a network structure. Localization is a fundamental problem in a sensor network. One example is a healthcare application in an assisted-living environment, where a healthcare WSN includes medical sensors worn by patients, such as EKG sensors [55], and other different medical sensors deployed in the infrastructure. Assuming patients are allowed to move around, it is necessary that such a healthcare WSN accurately reports the locations of patients in case that they have some medical emergency.

There are numerous applications that require not only location awareness of WSN nodes but, more specifically, the relative locations of WSN nodes [67, 52]. Let us consider *geographic routing* in WSNs. There are many different geographic routing algorithms, such as GPSR (Greedy Perimeter Stateless Routing) [43], GEDIR [53], GFG [12], and GOAFR [49], which are specifically designed to satisfy different applications of WSNs. All of them use sensor nodes' relative locations, at least, as their addresses, and then select a neighbor with the shortest projected distance. After that, these protocols forward packets in a greedy or other manner toward to the destination. In order to implement these protocols, we need to know the locations of nodes in their WSN.

Other examples include applications in security wireless sensor networks, which intend to protect WSNs against different attacks to disturb the network, such as wormhole attack. Localization can help to develop detection and defense mechanisms against such attacks by measuring irregular relative positions of nodes in such applications to detect those attacks

inside of the WSNs [34, 71, 97].

Fundamental to such seamless coordination in these applications is location awareness. Different applications need different granularity of location information. Geographic routing may only need the relative positions of nodes based on the shape of the network, while other applications such as healthcare applications may request WSNs to report physical locations of particular nodes as accurately as possible. Localization is a mechanism to establish spatial relationships in these wireless sensor nodes, and so plays a key role in providing such location service for different applications in WSNs.

1.2 Problem Statement

While current wireless sensor networks are applied to more and more applications, many applications involve both static nodes and mobile nodes. Even in pure mobile networks, such as in vehicle WSNs [35], some vehicles may park somewhere for a short time, so they may be considered as static nodes at that time. Also even in pure static networks, such as forest surveillance networks, some static nodes may fall from tree to ground, or some static nodes may be disabled then enabled because of power or environmental obstacles. Based on this case, we can safely say that, even in static networks, nodes still have some sort of mobility. So, localization in WSNs must consider the truth that WSNs are the networks, in which both static nodes and moving nodes are mixed at some time, and the number of moving nodes as well as the set of moving nodes is changeable. In this dissertation, we assume that localization in WSNs involves both static nodes and moving nodes at some points in time, and that the number of moving nodes as well as the set of moving nodes are changeable. We call such WSNs "mixed" WSNs. In mixed WSN, some nodes are moving while other nodes are static, and such status may change over time. Thus, a static WSN and a mobile WSN are two special cases of a mixed WSN.

WSNs are usually deployed or part of complex environments in different applications. For example, healthcare WSNs are deployed inside buildings; fire-alarm WSNs may be deployed in skyscrapers in municipal areas; and forest surveillance WSNs may be deployed in the wilderness. In order that such applications are not limited to open areas, where the Global Position System (GPS) can function, we are considering algorithms that do not rely on GPS. Also, GPS may not be affordable to be used by tiny sensor nodes, because of the high cost of GPS as well as the comparatively high energy consumption of a GPS module. In this dissertation, we focus on indoor environments where we do not relay on GPS for localization.

Another issue is that nodes inside a sensor network are intended to be guaranteed to work for a long time with limited energy. So, energy saving is also a primary consideration for localization WSNs. In a typical sensor node, the energy cost of communication is higher than that of computation, such as in a MICAz mote [37], receiving costs around 225% [37] and transmitting costs around 200% [37] of energy consumed by other components such as processors and sensor components in term of same time period. Especially in mixed WSNs, which involve some nodes in motion, updates of the locations of moving nodes will frequently be needed for localization over the lifetime of the networks, so related communication/computational costs will be high. How to efficiently keep the locations updated in the mixed WSNs will be a major issue in design.

Besides the energy constraint, memory and computational costs are important issues to be considered about running localization in WSNs. There is only ten kilobytes on-chip RAM memory in Tmote Sky/Invent [39] and XBow TelsoB [40] nodes and in general at best an 8 MIPS 8-bit processor in current sensor nodes.

Most current popular localization schemes are distributed to overcome the central-point-of-failure problem that exists in centralized schemes. But a large number of them require some set of anchor nodes, which already know their locations to compute the loca-

tions for unknown-location nodes. It is assumed that such anchor nodes do not fail and stay in a fixed location in their lifetime. However, in general, some of these nodes with prior location knowledge may fail, and this will hurt the accuracy of localization.

Especially in mixed WSNs, it is possible that some of those nodes with prior location knowledge (or anchor nodes) are non-static. With current technology, it is hard to keep the locations of nodes updated by them, especially in a non-GPS environment. So, if such nodes with prior location knowledge moved, the out-of-date prior location knowledge will significantly affect the accuracy of localization. The last argument implies that the localization schemes that rely on anchor nodes usually will need a large number of such nodes to guarantee that undiscovered nodes have enough anchor nodes nearby to calculate their locations. So, such scheme will be hurt if the number of anchor nodes with prior location knowledge decreases significantly or such prior location knowledge is out of date in a network.

Unfortunately, current localization algorithms only consider WSNs with static nodes, such as MDS-MAP [83], MDS-MAP(P) [82]. They only take into account the situation in which all the nodes inside the network are static. So, one possible way to apply these algorithms in mixed WSNs is to restart them frequently.

On the other side, several algorithms have been proposed on full mobile wireless sensor networks, in which all nodes in the network are kept moving. Such as MCL [31], MDB [4], they are relying on nodes with prior location knowledge to compute the locations for unknown nodes. The disadvantage for current solutions for mobile WSNs is that such solutions only consider the situation in which all the nodes inside a network are mobile. So, if we consider applying such methods in mixed WSNs, in which the assumption of all nodes moving is not always true, the algorithms, which dedicate to mobile WSNs, will face problems providing the energy-optimized, low computational cost localization service for mixed WSNs, especially when in the area where few nodes are moving. Also,

the accuracy of localization largely relies on the number of nodes with prior knowledge of location, but also depends on the mobility of the nodes.

From the above, it is clear that it is important to develop a localization system for mixed WSNs. Such localization should work well under condition of mixed WSNs that include both static nodes and mobile nodes.

1.3 Our Approach

Before we talk about our approach, let us define some concepts, which are to be used in this dissertation.

1.3.1 Definitions of Concepts

Definitions of Nodes

We consider sensor networks to be wireless sensor networks in this dissertation. Each node inside such networks deploys some low-power radio hardware such as TI/Chipcon CC2420 chip [27]. Each sensor network node also includes a battery pack as well as a set of sensors.

We define the hardware for a sensor node as a tuple of (S, P, W, B), where S is referred to as a set of Sensor(s) on the node; P is referred to as the microProcessor including memory and other necessary hardware; W is referred to as the radio for the Wireless communication; and B is referred to as the Battery.

Comparing with other Ad-hoc networks, such as laptop based Ad-hoc networks, the difference between sensor nodes (S, P, W, B) and laptop-level nodes is that the sensor nodes have different sensors in S, limited computational power and memory in P, limited energy in battery B, and a low-power short-range wireless device W.

Definitions of Mixed WSNs

Besides the definitions of nodes in WSNs, in this section, we talk about how to define WSNs. A WSN is an ad-hoc network consisting of spatially distributed autonomous devices including sensors to cooperatively monitor physical or environmental conditions. As we said previously, current WSNs – even some of them are defined as static networks – still involve some mobility. So, here we define WSNs based on their kinetic feature.

Definition 1: *mixed WSNs*: In a WSN with total n nodes, there are m nodes that are in motion at time t, while there are $n - m$ nodes in static state. We define such a WSN, which includes both static nodes and mobile nodes, as a mixed WSN. When $m = 0$, we call such a network a static WSN; when $m = n$, we call such a network a mobile WSN.

There are two special cases for the definition of mixed WSNs: when $m = 0$, they are considered as static networks, in which all nodes are static; when $m = n$, such a network is considered as mobile network, in which all nodes are in motion at the same time. While most of current assistive applications for WSNs [94] we are designing are in this range, in this dissertation, we only study mixed WSNs, where $m << n$.

Definition 2: (n, m) *mixed WSNs and* (n, m, a) *mixed WSNs*: In a mixed WSN with total n nodes, suppose that there are a nodes, which already know the prior knowledge of physical coordinates. We define such a network a (n, m, a) mixed network in our dissertation. In other words, in a (n, m, a) network, there are m (here $0 \leq m \leq n$) nodes, which are moving at time t, and a (here $0 \leq a \leq n$) nodes knew their locations already (also called as "anchor nodes"), no matter whether they are moved or not. When $a = 0$ in a (n, m, a) mixed WSNs, we can simply call such a mixed network a (n, m) mixed network.

Definitions of Localization

In general, localization can be treated as a graph reconstruction problem. Suppose there are total n nodes in a WSN, the graph of this network including all nodes is G and the set of the constraints for that set of nodes is C, we can say that a localization problem is to find a map of G' for all n nodes from the constraint set C. There are two types of localization algorithms based on whether a localization algorithm relies on anchor nodes, which are special nodes already know their locations before localization.

Definition of Anchor-Based (AB) Localization

Definition 3: Anchor-Based Localization: In a (n, m, a) mixed network with total n nodes, Anchor-Based Localization is to estimate the coordinates of $n - a$ nodes by using a set of $a > 0$ nodes with known physical coordinates (they are also called anchor nodes).

Definition of Anchor-Free (AF) Localization

Definition 4: Anchor-Free Localization: A $(n, m, 0)$ mixed network with n total nodes is considered. In such a network, Anchor-Free Localization is to estimate the coordinates of n nodes, without any special anchor nodes ($a = 0$), or in other words, without any nodes with prior knowledge of location.

1.3.2 Arguments against Anchor-Based Localization in Mixed WSNs

As we knew that there are two categories of localization: anchor-based and anchor-free. Here we talk about the reasons why we consider using anchor-based localization in mixed WSNs. We compare the two categories of localization under the background of mixed WSNs in the terms of the following features:

Accuracy

In an anchor-based localization, typically a large number of nodes are required to know their locations to achieve a reasonable precision in estimating other nodes' locations. For example, one anchor-based localization algorithm needs between 5% and 10% of all nodes [79], another algorithm needs between 10% and 20% of all nodes to know their locations, so that other nodes can rely on them to compute their own locations [80].

In a mixed WSN, any nodes, including anchor nodes, are intended to be moved. It is difficult to keep the locations of anchor nodes updated with current limitations in technology (such as GPS, which is unavailable in building). Since anchor-based localization relies on anchor nodes, when these nodes are moving or moved, it is hard for moving anchor nodes to be aware of their current locations exactly in practice, which decreases the accuracy of anchor-based localization.

Coverage

A node covered in localization means that a node is localized after localization algorithm finished [58]. It is possible that a localization algorithm can not generate the locations for all the nodes in a network. Anchor-based localization and anchor-free localization have different performance on coverage, since anchor-based localization largely relies on anchor nodes to compute the locations for other nodes nearby. Compared to anchor-free localization, anchor-based localization needs more anchor nodes to guarantee other nodes in the network to be covered (localized).

Security

Anchor-based localization assumes a trusted authority that can determine location information for anchor nodes (either through security hardware or through outside administration).

In environments where no trust relationships exist (e.g. in ad hoc networks created by proximity among strangers), this may be problematic. Both Anchor-based and anchor-free localization are vulnerable to attack, but in the anchor-based case there is no potential for attackers to self-assign special privileges, like the designation of "anchor node".

Efficiency

The last question is "efficiency", current anchor-based localization algorithms in mobile WSNs (for instance, MCL [31], ELA[92] and MCB [4]) require anchor nodes to broadcast their current locations periodically to help other mobile nodes update their locations. If some nodes are moving fast, then these localization algorithms will need to request anchor nodes to broadcast their locations more frequently to approach reasonable accuracy. It will have a significant communication cost for the localization over the entire network, even all nodes or some portion of the nodes keep static.

1.3.3 Our Approach

We try to address three design criteria for our approach:

- a distributed scheme
- a scheme that works with both static and mobile nodes
- accuracy, efficiency, and reliability

Horacio et al. make sure that current localization systems have three necessary components in general: distance estimation/measurement, position computation, and localization protocol [66]. When we design our approach, our approach includes three components: $M()$ (measurement): a component to measure distance between neighboring nodes; $C()$

(computation): a component to compute a local map for local area; $T()$ (transformation): a component to transform a local map into a portion of a global map.

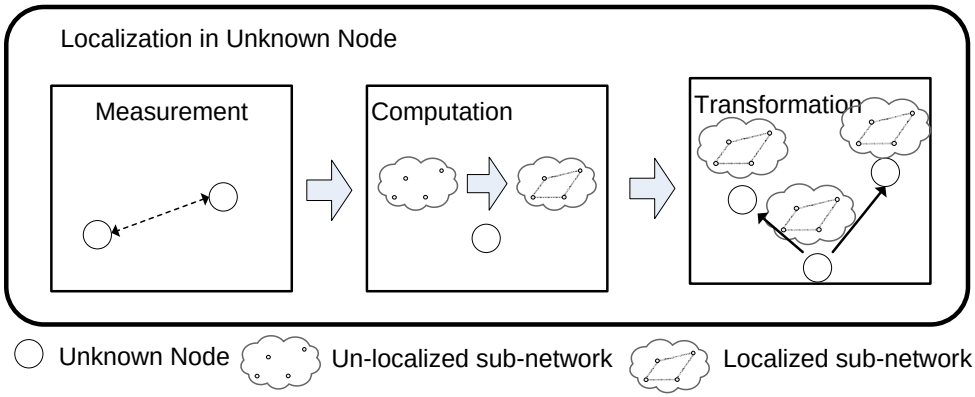


Figure 1.1: Anchor-Free localization in distributed scheme. Here, in the measurement component, each node measures distance $d.est$ to other node(s); in the computation component, each node computes a local map F, which is a set of estimated coordinates, for its neighboring nodes; in transformation component, each local map is transformed into a portion of a global map H node by node.

Both measurement component $M()$ and localization computation component $C()$ are necessary for a localization both for centralized and distributed scheme. When we consider a distributed version of localization approach over WSNs. Usually, the measurement component is done locally by low-cost low-performance sensors, which are deployed in all nodes in the network, to achieve the distance or angle measurement. After that, to compute a location for each node under one uniform coordinate system in a network, communication among nodes to convert the coordinates system will be necessary. So, we introduce an additional component called transformation $T()$ in our approach, so that each node can transform the location achieved locally to get a global map (in term of a 2D coordinate under a uniform coordinates system). A simple description is shown in Figure 1.1.

In Figure 1.1, we only consider the static case of (n, m, a) mixed WSNs when $m = 0$. When $m \neq 0$. It is possible that some nodes are moving upon time, so in our approach, we need a mechanism to update their locations for nodes with out-of-date locations. We define

"static" and "mobile" states for each node based on its kinetic status. To distinguish these two statuses, we introduce a sensor component $S()$ to detect the movement of each node. If a node is moved too much, we will then change the state of a node from static state to mobile state. At the same time, we will trigger a special localization procedure to update location for that node. Figure 1.2 shows the brief idea of how a sensor component triggers the transition of the states for a node.

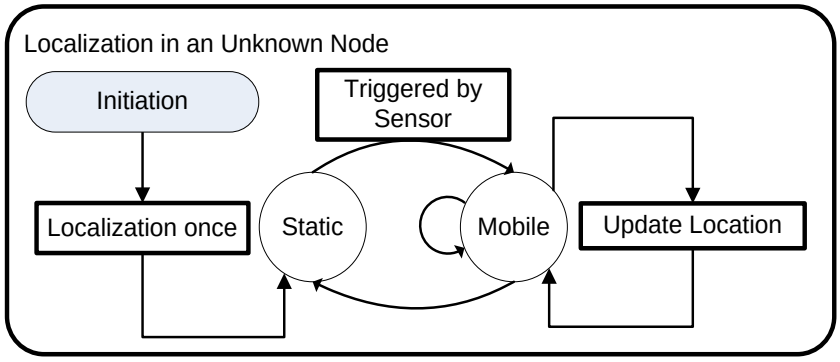


Figure 1.2: Anchor-Free localization with a sensor component for a node in mixed WSNs. The two ellipses represent two states (static and mobile) of a node. The node can stay in one of these two states. The change of a node's state is triggered by the sensor component that is shown as a rectangle in the figure.

How to provide an accurate location for each node is a primary challenge for localization. We implemented two novel measurement methods for our distributed localization. One is based on hop-counting, while the other is based on connectivity. Both require no additional hardware and achieve higher accuracy under same condition. We also design an Anchor-Free localization algorithm to test these measurement methods under static WSNs, which is a special case for mixed WSNs.

Because WSNs are resource-constrained networks, besides accuracy, efficiency is another consideration in designing our approach. In term of reducing communication and computational cost, we implemented our approach by integrating sensor components to detect movement of nodes. Our algorithm can achieve lower communication cost for Anchor-

Free localization in mixed WSNs.

Here, we consider reliability in term of prohibiting manipulated communication from localization. It is possible that normal communication of localization algorithms in a WSN is manipulated by some attackers. So, we proposed a distributed way to detect manipulated communication by wormhole when running a localization algorithm.

1.4 Thesis Contributions

Existing research work has laid the groundwork for localization in static WSNs, as well as for localization in mobile WSNs. However, the research work in static WSNs, which intend to solve the localization problem in static situation, can not be applied for the case of mixed WSNs without a significant increase in communication and computation costs. Prior research on mobile WSNs relies on the assumption that there are many nodes with prior location knowledge (anchor nodes) in the networks; this assumption results in higher communication and computation costs when applied to mixed WSNs.

Consequently, without help of nodes with prior location knowledge, none of the proposed systems can provide localization service for mixed WSNs in efficient, accurate and secure way. In order to address the localization problem in mixed WSNs, we have made several research contributions, which we list below:

- Contribution 1. The measurement component is important because it helps decide on the accuracy of localization both for static and mixed WSNs. We provide an accurate Range-Free measurement technique, called “hop-coordinates”, which can be a part of our localization algorithms. Compared to other methods in the same category, with the same data, our measurement technique has higher accuracy than other methods [60, 92] have. Chapter 3 discusses this work (published in the papers [100, 102]).

- Contribution 2. As a preliminary work of this thesis, a static localization is described for $(n, 0)$ mixed WSNs. Our localization for $(n, 0)$ mixed WSNs provides higher accuracy by combining with the measurement technique we developed, with constant memory cost as well as communication cost and computational cost comparing other algorithms. The details are presented in Chapter 4 (published in the paper [103]).

- Contribution 3. Efficiency is an important consideration for resource-constrained ad-hoc networks such as WSNs. A tradeoff between accuracy and cost is a key issue in designing localization for mixed WSNs. We provide localization in mixed WSNs that decreases the communication cost significantly without decreasing the accuracy by integrating sensor components, which detect the node's movement in mixed WSNs into localization algorithm, In addition, our method requires as few as possible additional measurement hardware or manual setup under both static and mobile network placements. The main idea is talked in Chapter 5 (published in the paper [104]).

- Contribution 4. Reliability is an important consideration for distributed localization in WSNs. It is possible that the accuracy of localization is affected by the manipulation of communication links in the WSN, such as is the case with wormhole attacks. Although wormhole attacks do not hack into the cryptographic infrastructure of the network, they have a large impact on the accuracy of different localization algorithms. To address this, we developed a distributed wormhole detection mechanism for our localization to detect such attack as well as to recover from the attack without relying on any centralized server or specific hardware. Details of this work is presented in Chapter 6 (published in the papers [101, 105]).

The dissertation concludes with limitation, discussions and future work in Chapter 7.

1.5 Symbols Used Throughout this Dissertation

In this section, we list the notation used in this dissertation.

Notation	Description
	the notation for wireless sensor networks
(n, m, a)	a (n, m, a) mixed network
N	set of total nodes
n	total number of nodes
M	set of nodes which are in motion (mobile nodes)
m	number of nodes which are in motion
A	set of anchor nodes
a	number of anchor nodes
	the notation for neighbors of a node
N_i	set of neighbor nodes of a node i
$I_{i,j}$	set of intersection nodes between the neighbor nodes set of node i and j
I_i	$I_i = \left\{\ldots, (x_k, y_k)', \ldots\right\}'$ (here $k \in N_i \cap I_{i,j}$, set of coordinates stored in node i
I_j	$I_j = \left\{\ldots, (x_k, y_k)', \ldots\right\}'$ (here $k \in N_j \cap I_{i,j}$, set of coordinates stored in node j.
	the notation for measurement
hop_i	the shortest hop distance for node i from some bootstrap node
$hop_{(i,j)}$	the shortest hop distance from node i to node j
offset_i	an offset measurement for node i based on some default bootstrap node
$hop.coor_i$	the smallest hop coordinates for node i from some bootstrap node
$hop.coor_{(i,j)}$	the smallest hop coordinates between node i and node j
$d_{(i,j)}$	the physical distance between node i and node j
$d_{(i)}$	the physical distance between node i and default bootstrap node
$d.est_{(i,j)}$	the estimated distance between node i and node j
$d.est_{(i)}$	the estimated distance between node i and default bootstrap node

Table 1.1: Symbols used throughout this dissertation

Notation	Description
	the notation for computation/transformation
t	time
T_i	transformation matrix for node i
T	transformation matrix in a node
x_i,y_i	2D coordinates of node i
c_i	physical location in 2D coordinate vector for node i, i = 1 ... n, and $c_i = \{x_i, y_i\}'$
f_j	estimated coordinate (x_i,y_i) for node j with whose coordinate system only covers the neighboring area of node i, here $j \in N_i$.
F_i	estimated coordinate matrix $F = [..., f_j, ...]$ in node i before it is transformed into H_i, here $j \in N_i \cup i$, also is called local map for node i
h_i	estimated coordinate for node i with whose coordinate system covers the whole WSN
H	estimated coordinate matrix for one uniform coordinate system in a network $H = [h_1, ..., h_n]$
$\vec{a}_i$	acceleration of node i at time t
$\vec{v}_i$	its velocity $\vec{v}$ at time t
d_i	the distance node i has moved at time t

Table 1.2: Symbols used throughout this dissertation (cont.)

Chapter 2

Related Work

In this chapter, we present the existing work that is relevant to the research performed in this dissertation. Measurement techniques for localization, localization algorithms, and attack-resilient localization in wireless sensor networks are discussed. In Section 2.1, we present work related to measurement techniques for localization. In Section 2.2, recent localization algorithms are extensively discussed. Related research in attack resilient localization is presented in Section 2.3.

2.1 Measurement Techniques for Localization

Localization measurement techniques can be roughly classified into two categories [22] based on different physical features: Range/angle-Based (RB) and Range-Free (RF) measurement technique. The term of RF is referred to the measurement techniques that do not rely on distance/angle measurements, but only utilize the connectivity information inside the network to measure the distance between nodes. In this section, we talk about connectivity-based and hop-counting measurement techniques in Section 2.1.2. Alternatively, RB type measurement techniques cover the techniques that try to get the physical

node-to-node distance or angle distance between sensor nodes, we talk about this category in Section 2.1.1.

2.1.1 Range/Angle-Based Techniques

There are several different popular RB techniques based on different hardware or different physical features. In this section, we talk about several popular RB techniques, which rely on physical measurements, such as signal strength and delivery time. We also discuss new emerging methods like Radio Interferometry (RI).

Signal Strength

Signal strength, or RSSI (Received Signal Strength Indication) in many contexts, is a measurement of received radio signal strength. In most of current implementations of WSNs, RSSI is measured by RSSI circuit based on received power integral. Usually, RSSI measurement consists of a one-byte value, which varies from 0 to 255 depending on different IC chip vendors. A value of "1" usually indicates the minimum signal strength detectable by a wireless receiver, while "0" indicates no signal.

In theory, signal strength is related to the physical distance from a source node to a target node. Suppose we know the transmit power p_t from the source node, that the environment is ideal vacuum, and that all other conditions are ideal. Then, theoretically, there is a perfect relationship between received signal power p_r and the distance d between the receiver and the transmitter given by:

$$\frac{p_r}{4\pi d^2} = \frac{p_t}{4\pi 1^2} = \frac{p_t}{4\pi} \Rightarrow d = \sqrt{\frac{p_r}{p_t}} \tag{2.1}$$

If we know the transmit power p_t and the received signal power p_r, then, with the above equation, we can compute the distance between the source node and the target node based

on the RSSI measured in the target node and the predefined transmit power from the source node.

RSSI measurement is relatively inexpensive and simple to be implemented in hardware. Currently most of the hardware implementations of WSNs support it. It is an important and popular topic of localization research in a long time. But, in practice, the relationship between physical distance and RSSI is affected unpredictably by the antenna, battery, electric components inside of the node, and outside environment [107]. They make RSSI measurement to be notoriously unpredictable [48, 107].

AOA (Angle-of-Arrival)

AOA (Angle-of-Arrival) measures local angle information to neighboring nodes, which can either be used as complementary to other distance measurements (such as RSSI) [63] , or be used to compute the locations of nodes [14] with the help of connectivity information, which can be achieved in any WSNs.

Currently, there are several ways to implement AOA measurement. One method published in [63] is to achieve local angle information by using directional antennas. By deploying directional antennas to all nodes in a network, a node can detect the local direction to its neighboring nodes roughly if the direction of the directional antennas in that node and its neighboring nodes is known. Another way is to deploy several transceivers with specific angles in each node. Each transceiver is then used to establish connections with its neighboring nodes using the known angles to compute the local angles from this node to each neighbor. One implementation is proposed in the cricket system [74]. In another implementation in [74], angles between adjacent edges are measured by using multiple ultrasound receivers.

The problem for AOA technique in current implementations is that current implementation techniques require either directional antennas, which need manual deployment to

guarantee adequate coverage, or multiple transceivers. Each of these options leads to additional cost as well as deployment difficulty.

TOA (Time of Arrival)

Time of Arrival (TOA) measures the time at which a signal (such as a radio, acoustic, or other types of signals) first arrives at a receiver. The measured TOA is the time of transmission plus a propagation-induced time delay. Since propagation-induced time delay has a linear relationship to the distance between the receiver and the transmitter, it is possible for TOA measurement technique to use TOA to compute the distance between a receiver and a transmitter.

For example, suppose there are a transmitter node i, a receiver node j, and a TOA $t_{i,j}$ from node i to node j for one packet transmission. Both transmitter and receiver are radio frequency devices and the propagation velocity for Radio is v. Then, we can compute the distance between node i and node j as follows:

$$d = v \times (t_{i,j} - C) \tag{2.2}$$

Here, C is the transmission time of a packet, and if the length of that packet is constant, we can consider the transmission time of that packet to be constant for a particular node.

One simple approach to implement TOA measurement is to deploy an acoustic transceiver, which has much lower propagation velocity than RF signal, for each node. TOA measurement works as follows: A node sends a message, which includes the current time stamp, out. Another node, which receives the message, can use that time stamp to compute the distance to the sender based on TOA.

Since TOA measurement needs to share a global time in a WSN to compute the TOA as well as the distance, it requires an accurate time synchronization protocol among nodes.

But with current low-cost implementations of WSNs, a high accurate time synchronization protocol is difficult to be achieved. A popular clock synchronization algorithm [80] in WSNs has reported synchronization precision to the order of 10 μs. Such synchronization accuracy is adequate for TOA measurement based on acoustic signals [26], but it is not enough for TOA measurement based on RF signals, because of the significant difference between the signal propagation speeds.

For TOA measurement in asynchronous sensor networks, one common practice is to use two-way (or round-trip) TOA measurement. But in this case, more communication costs will be involved in round-trip TOA measurement.

TDOA (Time Difference Of Arrival)

TOA measurement needs high-accuracy synchronization as well as lower propagation-speed transceivers, and it also increases the propagation time of normal messages. TDOA (Time Difference Of Arrival) measures the time difference of arrival of two different signals having different propagation velocities. It avoids the complex synchronization, which may be required by TOA.

In wireless sensor networks, the signal used to transmit the message is radio wave, whose propagation speed equals the speed of light. So compared with the transmission time, the propagation time will be the last thing to be considered when distances are short. In reality, one approach is to deploy two different transceivers in each node, such that one is an RF transceiver, the other is an acoustic transceiver. So the difference between propagation speeds of the two different signals is significantly large. Then, let the source node send out same message at the same time with both transceivers. The target node computes the difference between the two TOAs for the two messages, and uses it to compute the distance between the source node and the target node. Suppose node i is the source node, node j is the target node, and $t1_{i,j}$ is the TOA from node i to node j for one signal transmitter (an RF

transmitter with propagation velocity v_1). Let $t2_{i,j}$ be the TOA from node i to node j for another transceiver (an acoustic transceiver) with propagation velocity v_2. We can compute the distance between node i and node j as follows:

$$d = (v_1 - v_2) \times (t2_{i,j} - t1_{i,j}) \tag{2.3}$$

One implementation of TDOA measurement is the Cricket location-support system developed at MIT, which is proposed in [74]. In this system, sensor nodes called crickets are equipped with a microprocessor, a Radio Frequency (RF) transceiver, and an ultrasonic transceiver. Inter-node distances can be measured at the accuracy of approximately one centimeter by measuring TDOA between an ultrasound pulse and RF pulse.

Radio Interferometry (RI)

There is a new technique called Radio Interferometry (RI), which was first proposed by Maroti et al. [56]. The novel idea behind RI is to utilize two transmitters to create an interference signal directly. Two transmitters in two transmitter nodes send out the RF signal with almost same but not exactly same frequencies f_A and f_B, which is easy to be achieved in current WSN transmitters. Then, the composite will have a low frequency envelope $|f_A - f_B|$, which can be measured by cheap hardware available in WSN nodes. In order to avoid high-cost synchronization, they proposed the use of two receiver nodes to receive that frequency envelope at the same time. Then by comparing the two phases in this frequency envelop in the two receiver nodes, a phase offset can be computed. The phase offset is a function of the relative positions of these four nodes (two receiver nodes and two transmitter nodes). This function is described in the following Figure 2.1. By making multiple measurements in an at least 8-node network, it is easy to reconstruct the relative locations of nodes in that network.

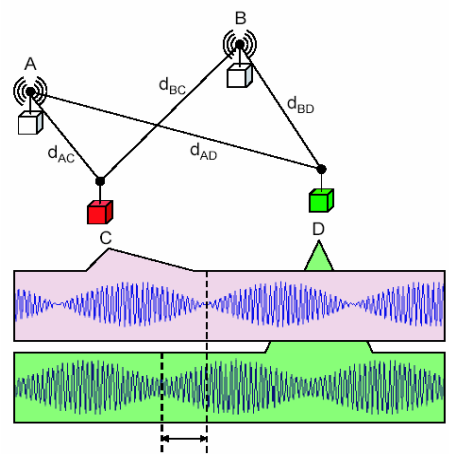


Figure 2.1: Phase offset in Radio Interferometry in 4 nodes (The picture is directly copied from [56]).

If

$$phase_D = \frac{2\pi \left| d_{AD} - d_{BD} \right|}{\lambda_{|f_A - f_B|}} (\mathrm{mod} 2\pi) \tag{2.4}$$

then

$$phase_C = \frac{2\pi \left| d_{AC} - d_{BC} \right|}{\lambda_{|f_A - f_B|}} (\mathrm{mod} 2\pi) \tag{2.5}$$

and so phase offset, which is shown as arrowed line in Figure 2.1, as following:

$$phase\_offset = \frac{2\pi(|d_{AD} - d_{BD} + d_{AC} - d_{BC}|)}{\lambda_{|f_A - f_B|}} (\mathrm{mod} 2\pi) \tag{2.6}$$

RI is a very exciting measurement technique, which achieves very high accuracy without any additional hardware. Two small problems involving in RI are: 1. The technique requires synchronization in neighboring nodes; 2. The distance that can be measured is limited by the wavelength of the frequency envelop.

Other Emerging Techniques

Besides the popular measurement techniques mentioned above, more and more techniques are emerging. To the best of our knowledge, from our most recent reading, all of these

methods either have limited capability in some features, or have low potential for large-scale utilization in resource-constrained WSNs. In this section, we mention some techniques that are different from the techniques mentioned in previous sections.

One of the most recent work is light-based localization [88]. This technique uses a strong light source from outside a WSN to scan the whole physical area the WSN is deployed to localize the nodes in the WSN. This method achieves very high accuracy in localization, but the shortcoming is that it needs a powerful external light device and a precise mechanical system to guarantee the quality of scanning.

Even though the techniques in this category are immature or can only be achieved by special devices, or are limited into special localization applications, we cannot deny that it is possible to evolve an accurate approach without many costs in future based on these techniques.

2.1.2 Range-Free Techniques

In general, RB techniques usually achieve more accurate localization results, but at the same time, they need additional hardware or special deployment methods for distance/angle measurement. Range-Free (RF) techniques that do not need any additional hardware or special deployment for the measurement are introduced in this section.

Connectivity

Connectivity is the information that is used to represent whether a node is connected with other node(s) or not. Connectivity-based measurement technique uses only connectivity information to measure the distance between different nodes. In theory, if we just consider connectivity, a multi-hop radio network can be modeled as a unit disk graph (UDG). In UDG modeling, all nodes in WSNs have the same transmission range. Two nodes can

communicate directly with each other when they are within each other's transmission range.

Let i be transmitter node, j a receiver node, and $\widehat{d_{ij}}$ the real distance from node i to node j. With all wireless nodes having the same transmission range $\widehat{d_{\max}}$, the distance between node i and node j can be computed based on connectivity, as follows:

$$d_{ij} = \left\{ \begin{array}{c} 1 : \widehat{d_{ij}} \leq \widehat{d_{\max}} \\ \infty : \widehat{d_{ij}} > \widehat{d_{\max}} \end{array} \right\} \tag{2.7}$$

Some more detail about the implementations can be found in papers [10, 65].

From the above formula, we can see that compared to the range-based measurement techniques, the connectivity-based measurement technique provides less accuracy in measurement. However, the connectivity-based measurement technique has an inherent advantage. Since connectivity information is naturally represented by the connection relationship in sensor networks, connectivity-based measurement can be achieved at low cost and without additional hardware. The technique can be implemented in low-cost WSNs or can be made to work complementary to other measurement techniques.

Hop-Counting

Hop-counting is another Range-Free technique that needs no special device to do measurement. In order to do hop-counting, a pre-appointed bootstrap node is needed, that sends out a hop-counting message with a variable *hops* = 0. Using the initial message, every other node determines its hop distance from the bootstrap node and forwards the message after accumulating the variable *hops*.

Consider a node a that wishes to calculate the hop distance, and node b is one of its neighbors. Then, the basic hop-counting procedure for node a can be shown as follows Procedure 1:

Here, a is a node, hop_a and hop_b represent the minimum number of hops to reach node

Procedure 1 Hop counting procedure in node a

1: INPUT: message (hop_b) from node $b \in N_a$
2: $hop_a = \infty$
3: **for** message (hop_b) from any b $\in N_a$ and not TIMEOUT **do**
4: **if** $hop_b < hop_a$ **then**
5: $hop_a = hop_b + 1$
6: transmit(message(hop_b +1)) {makes change to hop_b in the received message, then forwards it to the neighboring nodes}
7: **else**
8: drop(message(hop_b))
9: **end if**
10: **end for**
11: RETURN hop_a

a and b respectively, counting from some bootstrap node x. N_a is a set of neighboring nodes, which can be reached by node a in k hops, and $|N_a|$ is the number of nodes in N_a.

DV-HOP [64] uses a technique based on distance vector routing. Each node maintains a counter denoting the minimum number of hops to each bootstrap node, and updates that counter based on messages received. Each bootstrap node propagates hop-counting messages through the network. When a node receives a new hop-counting message from a bootstrap node, it updates its hop count to the new value if its hop count is lower than the hop count in the message. The node then retransmits the hop-counting message with an incremented hop count value.

The Gradient localization algorithm [60] uses a similar approach. In this algorithm, the bootstrap node works as an anchor node. The coordinates of bootstrap nodes are flooded throughout the network so that every node can maintain a hop-count to the anchor nodes. Nodes calculate their position based on the locations of bootstrap nodes and the result of the corresponding hop count timing transmission range.

2.1.3 Comparison of Two Categories

As a key component of localization algorithm, measurement techniques only address the problem of how to measure the distance/angle between each pair of nodes. As discussed above, a range-based measurement technique usually achieves more accurate results than a range-free measurement does. But, range-based measurement may need additional hardware or special deployment or complex synchronization. On the other hand, there is no conflict between range-based and range-free measurement techniques, research in both categories is still valuable as they can be combined together to deduce total localization error. In fact, many localization algorithms already combined these two techniques. Some implementations combine AOA and connectivity information in the localization algorithms [14, 63].

2.2 Localization Algorithms

The last section discussed measurement techniques for localization. After measurement, the next step is how to compute location for each node based on measurement. This section discusses work related to the computation part in localization algorithms.

2.2.1 Anchor-Based and Anchor-Free Localization

We classify localization algorithms with the truth of whether a localization algorithm uses nodes with prior location knowledge. These nodes, called anchor nodes or reference nodes in some other papers, are special nodes that already know their physical coordinates before localization starts. We use the term localization without prior knowledge (or Anchor Free (AF) localization in other papers) to refer to the localization algorithms that do not use specially designated reference nodes with known physical coordinates. The alternate to

localization without prior knowledge is localization with prior knowledge or Anchor-Based (AB) localization, which relies on some nodes with prior knowledge of their locations.

Anchor-Based Localization

As a special node, an anchor node usually uses some expensive localization device such as GPS, or is located in advanced. The basic idea of anchor-based localization is: i). Each node, except anchor nodes, measures its distance to the anchor nodes with the measurement techniques mentioned in the previous section; ii). After an un-located node measures its distance to enough number of anchor nodes, it computes its coordinates in physical space with some localization algorithms. In order to get two-dimensional coordinates for a node, at least three anchor nodes that are not in the same line are needed for the measurement to determine a node in a coordinate system. In three-dimension space, at least four anchor nodes that are not in the same plane are required.

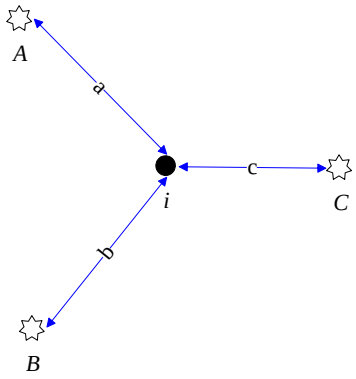


Figure 2.2: Anchor-Based localization. Here, nodes A, B and C are three anchor nodes, which know their locations, and node i is a node to be localized. Node i measures the distances a, b and c to nodes A, B and C, separately.

In Figure 2.2, for example, three anchor nodes A, B and C that are reachable from node i, can be used to locate a node i with coordinates (x, y) in two-dimensional space. The anchor nodes have coordinates are $((x_A, y_A)$, (x_B, y_B), $(x_C, y_C))$, respectively. The

distances measured from node i to anchor nodes A, B, C, are a, b, c, respectively Then the following equations can be used to compute the coordinates of node i:

$$\begin{cases} a^2 = (x_A - x)^2 + (y_A - y)^2 \\ b^2 = (x_B - x)^2 + (y_B - y)^2 \\ c^2 = (x_C - x)^2 + (y_C - y)^2 \end{cases} \tag{2.8}$$

Some representative algorithms in this category include [30, 51, 74, 69, 76, 79, 63, 88], Centroid [19], Cricket [59], AHLoS [80], RADAR [5], APIT [29], Gradient [60], APS [62], DV-hop [64] and P-Grid [16]. They are either using Range-Based or Range-Free measurement techniques.

Anchor-Free Localization

Anchor-Free localization does not need to rely on anchor nodes neither at the time it is measures the distance between nodes, nor at the time when it computes the node coordinates.

Usually, anchor-free localization has two steps to compute the relative coordinates of nodes: i). Each node measures the distance to its neighboring nodes with either Range-Free or Range-Based measurement techniques mentioned before. ii). After measuring the distance from a node to its neighboring nodes, the node computes the relative coordinates for itself and its neighboring nodes, even if the physical coordinates of the other nodes are not known. [1]

Rao et al. proposed an algorithm for two-dimensional WSNs, which considers two different situations without anchor nodes: one where perimeter nodes are known, and another

[1] Anchor-free localization only generates relative coordinates for each node. To identify physical coordinates, an additional step is needed: After the relative coordinates of the nodes in a network are calculated, two-dimensional coordinates can be calculated if three anchor nodes can be found; for three-dimensional coordinates, four anchor nodes are needed.

where perimeter nodes are unknown [75]. The authors showed that both cases can actually be combined into the first case, as their algorithm can determine the position of perimeter nodes in a WSN. Once perimeter nodes are identified, perimeter and bootstrap nodes broadcast specific messages. Because the perimeter of a WSN is usually related to the number of nodes, we can see from Table 1 in paper [75] that the communication cost will be roughly $O(n^{1.5})$, where n is the number of nodes in the WSN. Since our algorithm, which is to talk in Section 4 and 5, does not need to broadcast many messages from perimeter nodes, there will typically need to be only $O(n^{0.5})$ perimeter nodes, allowing savings on communication costs, which can be limited to $O(Cn)$ (here C is a constant bigger than 1) for the overall network, and $O(1)$ per node. For large-size WSNs of constant density, the number of nodes in any local area is limited by a constant C, which is much smaller than n.

Shang et al. described an algorithm called MDS-MAP(P) [82] that extends MDS-MAP [83] into a distributed algorithm. MDS-MAP(P) first computes local maps for each node with MDS using local shortest paths, then merges local maps into a global map. MDS-MAP(P) outperforms most other algorithms and has a computation cost of $O(k^3 n)$ (here k is the number of neighbor nodes). Even if the algorithm does not include its optional refining step, it still needs at least $O(n \log(n))$ communication cost based on using a binary aggregation tree to route messages to send back the local map from each local node.

Another drawback of MDS-MAP(P) is that the memory requirement per node will be $O(n)$ to store the $O(n)$ global map when the local map merging process is being completed. In our algorithm proposed in Section 5, in contrast, we require only $O(1)$ memory per node.

2.2.2 Range-Based and Range-Free Localization

Besides the above two categories, orthogonally, we can classify localization algorithms with different measurement techniques by using the categories discussed in Section 2.1.

Range-Based Localization

	Range-Based technique						
	*Unspecified distance	AOA	TOA	TDOA	RSSI	RI	others
Anchor-Based scheme	[73], [79]	[51], [63], [74]	Centroid [19], [30]	Cricket [59], AHLoS [80],	RADAR [5], [68], [69]		[76], [88]
Anchor-Free scheme	MDS-MAP [83], MDS-MAP(P) [82], [8], [81]	[14]		Cricket [72]		RIPS [56], [70]	

Table 2.1: Localization algorithms with Range-based techniques
*In these papers, they are using distance to do localization, but did not mention what kind of distance measurement techniques they are using, so, here we separate them into a single category. But theoretically, such algorithms can utilize TOA, TDOA and RSSI measurement techniques.

Range-based category uses absolute point-to-point distance or angle information between neighboring sensors to calculate the location for each node. One common technique for distance/angle estimation is to use angle of arrival (AOA) in addition to hop information [14, 63]; similar methods use time of arrival (TOA) [19, 30] or time difference of arrival (TDOA) [80, 72], or Received Signal Strength Indicator (RSSI) [107] in addition to hop information.

As noted above, AOA, TOA, and TDOA all have additional hardware requirements beyond what is needed for basic WSN functions. AOA requires additional devices such as ultrasound transceivers [74] or directional antennas [62], and TOA and TDOA requires nodes to have two different communication devices with unequal propagation speeds, such as ultrasound transceivers and RF transceivers [74], to measure time differences. The utilization of RSSI depends on a direct relationship between the distance and the signal strength, but

this relationship is affected unpredictably by the antenna, battery, electric components inside of the node, and the outside environment [107]. In summary, range-based protocols are frequently cost-ineffective as alternatives for producing fine-grained locations due to the requirement of additional hardware, the strict requirements on time synchronization this hardware entails, and the resulting increase in energy consumption.

Range-Free Localization

	Range-free technique	
	Connectivity	hop-counting
Anchor-Based scheme	APIT [29], [78]	Gradient [60], APS [62], DV-hop [64], P-Grid [16], [10]
Anchor-Free scheme	MDS-MAP [83], MDS-MAP(P) [82], [81]	

Table 2.2: Localization algorithms with Range-Free techniques

Typical approaches to avoiding range hardware, such as those described in [10, 29, 60, 75, 79, 82, 83], make use of the connection information of the network—something inherent in any WSN without any additional hardware. Some approaches are based on the idea of letting nodes derive their position in terms of connectivity to special anchor nodes (which have predetermined geographic information) [29, 60, 79]. While other approaches are not relying on the anchor nodes, such as [82, 83].

2.2.3 Mobile Localization

The localization algorithms mentioned in previous section assume that the nodes inside a network are static, while the applications of wireless sensor networks are not limited

in static [86]. When considering localization in mixed networks, current research groups assume that the anchor nodes will always keep their location knowledge updated, and so other nodes without location, can use multi-iteration type algorithms such as Karmen Filter, Monte Carlo [4, 31] or Neural Networks [2] to update as well as increase the accuracy of location.

S. Čapkun et al. proposed an Anchor-Free localization called SPA for Mobile WSNs, which localizes nodes in mobile sensor networks through triangulation of neighbor nodes [19]. SPA first computes a relative coordinate system for each node, then converts the above coordinate system in each node into a global coordinate system by calculating differences in terms of distance and direction between each node and a particular central node, or a dense group of nodes called Location Reference Group (LRG). The problem for SPA is that if any nodes, especially those nodes inside an LRG, are moved, then a recalculation must be done to almost the whole network, which is costly and unnecessary.

Hu and Evans present a range-free Anchor-Based localization algorithm [31] for mobile sensor networks based on the Sequential Monte Carlo method [16]. The Monte Carlo method has been extensively used in robotics [5] where a robot estimates its localization based on its motion, perception and possibly a pre-learned map of its environment. Hu and Evans extend the Monte Carlo method as used in robotics to support the localization of sensors in unmapped terrain. The authors assume a sensor has little control and knowledge over its movement, in contrast to a robot. A similar paper [4] shares the same idea.

By using an analogy with a system of springs and masses, the Elastic Localization Algorithm (ELA) [92], as an anchor-based algorithm, tries to calculate locations with anchor nodes, which already know their locations. A mobile version of ELA supports mobile localization by updating neighbors' locations in each node at fixed intervals. Such periodic updating in the whole network may lead to huge communication and computation costs.

Akcan et al. propose an anchor-free method that focuses on locating group move-

ment [36] — cases in which multiple nodes have a similar direction and velocity. By deploying a compass in each node to detect the direction of a node, each node computes the relative locations of its neighbors. Their work pays particular attention to group movement, and does not consider independent movements by individual nodes, which is an important and common case in mobile networks.

2.2.4 Summary

Since a large number of different localization algorithms have been proposed, we list a representative set of localization algorithms based on categorization as Range-Free or Range-Based, discussed in the previous section. Table 2.1 and Table 2.2 show the localization algorithms using Range-Free techniques and Range-Based measurement techniques, respectively. In both tables, the rows are categorized based on Anchor-Free and Anchor-Based computation methods. We also talk mobile localization algorithms in Section 2.2.3.

2.3 Attack Resilient Localization

Localization is a fundamental service in WSNs that provides necessary support for many important WSN protocols, such as geographic routing and data-fusing protocols. If a WSN is deployed in a hostile environment, localization can be a target for possible attacks that arm to disable some functionality of a sensor network. There are many different kinds of attacks against WSNs, such as, wormhole attacks, Sybil attacks [61], and jamming and packet injection attacks [99] against WSNs. Amongst these, wormhole attacks [18, 34] are a threat that do not require knowledge of the cryptographic infrastructure of a network. In this section, we discuss related work on how to detect wormhole attacks in localization and defenses against such attacks.

2.3.1 The Wormhole Attack

In physics, a wormhole is a hypothetical topological feature of space-time that is essentially a "shortcut" through space and time. A wormhole is an adversarial structure that disturbs the spatiotemporal relationship in multi-hop communication in WSNs.

In a typical wormhole attack, a wormhole receives packets at one point in the network and forwards them through a wireless or wired link referred to as a "tunnel". The tunnel has much lower latency than regular network links and quickly relays the packets to another position in the network. Here, we assume that a wormhole is bi-directional and comes with two endpoints called "ends", although multi-end wormholes are theoretically possible. Figure 2.3 shows a typical wormhole attack.

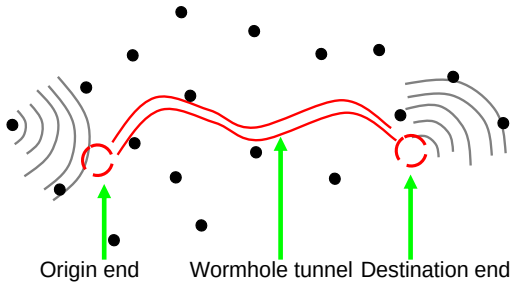


Figure 2.3: A wormhole attack in a WSN. Here, the figure presents an ad hoc network of 20 nodes and a wormhole link between left and right areas of the network. When node 1 in the left area sends out a message, then this message is tunneled through the wormhole link, nodes in the right area will hear the broadcast and assume that node 1 is only a one-hop away.

Since a wormhole attack does not rely on knowing the cryptographic infrastructure of the sensor network, an attacker can carry out this localization attack in a sensor network even if the network communication infrastructure provides authentication mechanism and even if the attacker does not have any cryptographic keys.

2.3.2 Wormhole Detection

Some statistical techniques have been proposed to detect wormholes in networks, examples include Haystack [85] and NIDES/STAT [41]. Other methods include approaches based on machine learning methods: IBL [50], TIM [89], computer immunological-based methods [24, 25, 98], and some other methods described in [45, 46, 95]. But because of the computational constraint and resource constraint on sensor nodes, most of these techniques cannot be directly applied to sensor networks.

Wormhole attack detection in wireless ad-hoc networks or WSNs was introduced in several papers [18, 33, 34]. One solution to address this problem is referred to as "Packet Leashes" [33, 34]. Čapkun et al. describes an approach called SECTOR [18]. These mechanisms detect wormhole attacks based on the notion of geographical or temporal leashes. Briefly, suppose that every node in the network already knows its exact location and each node embeds its location and timestamp into every packet that it sends. If such a network is synchronized, then other nodes in the network that receive that packet can detect a wormhole by noting the mismatch between the timestamp difference they calculate and the location difference they observe.

Kong et al. study Denial of Service (DoS) attacks including wormhole attacks, in UWSN (Under Water Sensor Networking) [47]. Since UWSN typically uses acoustical methods to propagate messages under water, the methods in UWSN cannot be directly applied to wireless sensor networks.

Poovendran et al. present a useful graph theoretic framework for modeling wormhole attacks [71]. However, the theoretic framework is based on the assumption that there are "guard nodes" that know their exact locations. Thus, these nodes actually work as anchor nodes as described in dissertation. Since in this dissertation we assume that none of the nodes in the network knows its physical location, which is more reasonable than Pooven-

dran et al.'s all nodes knowing their physical locations, our proposed solution is for a case not covered by this framework.

MDS-VOW [97] allows visualization of a network to detect wormholes by finding bending distortions caused by a wormhole in a computed map. One disadvantage of MDS-VOW is that it can only work in a centralized scheme, so MDS-VOW needs to have a central computer to finish its computation. In our algorithm described in Section 6, we extract a new feature which can efficiently indicate the ends of a wormhole based only on local bending distortions caused by the ends of the wormhole. The algorithm described in Section 6 is computed by a distributed scheme and requires no centralized computation. Another limitation of MDS-VOW, which is identified by Poovendran et al., is that such a visualization cannot be applied to networks with irregular shapes [71]. An example of a network with irregular ship is a string topology wherein nodes are connected in one line.

2.3.3 Wormhole Resilient Localization

Besides the idea to detect wormhole in WSNs, another idea is to integrate wormhole detection and defense mechanism into localization. This is called as "Wormhole Resilient Localization". Wormhole resilient localization detects wormhole attack during or before localization, and tries to decrease the effects of the attacks in the localization results.

Hu and Evans utilize directional antennas to prevent wormhole links by assuming every node of the network will be equipped with directional antennas that all have the same orientation [32]. Lazos and Poovendran apply a similar idea in designing a secure localization scheme called SeRLoc [51] that protects against wormhole attacks in localization. In SeRLoc, there are about 400 anchor nodes (designated as "beacon nodes" in the paper) deployed in a 5000-node network. Each anchor node has a directional antenna and already knows its physical location. Other nodes in the network use these anchor nodes to

locate themselves. During a wormhole attack, a wormhole shortcuts the network and the directional antennas deployed in the anchor nodes help detect the attack. The network can then defend themselves against the attacks by discarding incorrect localization message. However, if anchor nodes are compromised, especially those anchor nodes that are close to a wormhole end, SeRLoc will have difficulty in detecting/defending against the attacks. Another issue is that the large number of anchor nodes SeRLoc uses require significant manual setup.

In more recent papers, D. Liu et al. proposed an anchor-based scheme, which is resistant to several different types of attacks, including wormhole attacks [22, 54]. By using a hop-counting technique, the scheme estimates the distance between a node and an anchor node (or "location reference" in the authors' terminology). If there is a wormhole inside the network, then it is possible that the hop distance from a node to some anchor node will be changed. A simple threshold method is used to determine whether such a distance difference is caused by a wormhole attack or by localization error. The disadvantage of this method is that it relies on anchor nodes that need to be setup manually in advance. The main difference between our method (proposed in Section 6) and those of [22] and [54] is that the latter methods rely on anchor nodes, which need manual setup in advance, while our method does not require any anchor nodes to detect wormholes.

All of the above methods use anchor nodes as an important feature to defend against the wormhole attack. This is another reason why these methods are difficult to be incorporated into anchor-free localization schemes, which are more flexible than anchor-based localization schemes.

Chapter 3

Improving Measurement Accuracy

In this chapter, we talk about a necessary component in localization for WSNs – measurement. Measurement component in localization tries to measure the distance between each pair of nodes, so that, the measured distances can be used by the computation component in localization, which is to be introduced in later chapters, to calculate the location for each node in the network.

While accuracy is an important consideration for localization system in a mixed WSN, we propose a new abstraction for measurement in localization — hop coordinates, to improve the accuracy of measurement. This method works for both localization systems with static nodes and mobile nodes (which we will talk in Chapter 4, 5).

3.1 Introduction

There are many physical features in WSN that have been utilized by different localization algorithms to generate the locations of nodes in a WSN: the Angle Of Arrival (AOA) [63], Time Of Arrival (TOA) [26], Time Difference Of Arrival (TDOA) [74], Receive Signal Strength Indicator (RSSI) [107], Radio Interferometry (RI) [56] and hop information and

connectivity. Usually, sensor nodes must deploy additional hardware for AOA, TOA and TDOA, or special antennas for AOA. As for RSSI, the utilization of RSSI depends on a rough relationship between the distance and the signal strength, but this relationship is affected unpredictably [107] by the antenna, battery, electric components inside of the node and the environment outside of the node. In contrast to other physical features that we can use in localization algorithms, hop information and connectivity are much more stable and can be achieved in most currently deployed WSNs without any additional hardware.

Extensive simulation experiments on different localization algorithms based on hop-counting or connectivity alone have usually shown a sizable error between the actual location and the estimated location [10, 79, 82, 83]. This distance error persists even after applying some refinement procedures [82, 83]. While approximate localization is acceptable for a number of purposes, including routing, increasing the accuracy of localization would be beneficial for many applications.

This chapter does not propose a completely new localization algorithm; rather, based on hop-counting it introduces a novel abstraction called hop coordinates. This technique not only counts the number of hops, but also offsets the total based on the local network structure. Such an approximation can be used on any localization algorithms to which hop-counting or connectivity techniques can apply.

3.2 Inaccuracy of Current Measurement Techniques

In this section, we introduce two measurement techniques briefly, and describe the causes of the inaccuracy at current measurement techniques.

3.2.1 Hop-Counting Technique

Hop-counting technique needs no special hardware to calculate localization information. Hop-counting is based on having a pre-appointed but arbitrary bootstrap node send out a hop-counting message with variable *hops* = 0 inside. By using that message, each node determines its hop distance from the bootstrap node and forwards that message after increasing the variable *hops*. This is discussed in detail in related research work in Chapter 2.

3.2.2 Problems with Hop-Counting

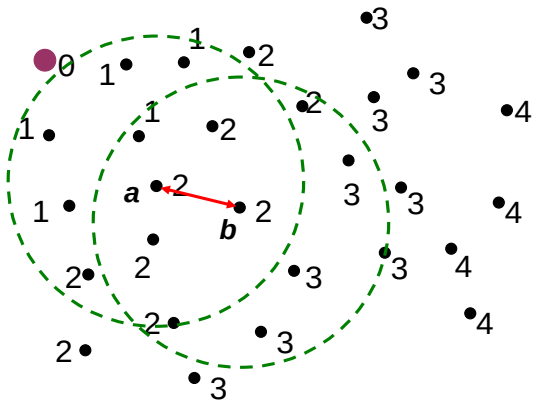


Figure 3.1: An example of two nodes with the same hop count but different distance to a bootstrap node. In the figure, node a and node b share same number of hops ("hop counts") to a bootstrap node, despite being significantly far apart (the distance between node a and node b is shown in an arrow). The dots represent nodes in the network, the oversized node is the bootstrap node, and the number near each node is the hop count for that node. Circles around a and b indicate the radio ranges for node a and b.

The basic idea behind hop-counting is that there is a special bootstrap node in the network, which sends out a message to flood the network, and all other nodes in the network will use this message to count the number of hops to that bootstrap node. This is an efficient way to measure the distance between any two arbitrary nodes in wireless sensor networks.

However, the problem is that this method only distinguishes nodes based on the number of hops between them, so if many nodes share the same number of hops distance from the bootstrap node, they cannot be differentiated. One example is shown in Figure 3.1.

From Figure 3.1, we can see that nodes a and b are identified as having the same hop number based on the number of distinct hops it takes to reach them from the bootstrap node. In actuality, nodes a and b have different physical distances to the bootstrap node. Unfortunately, this problem occurs frequently in a typical network, as illustrated in Figure 3.2.

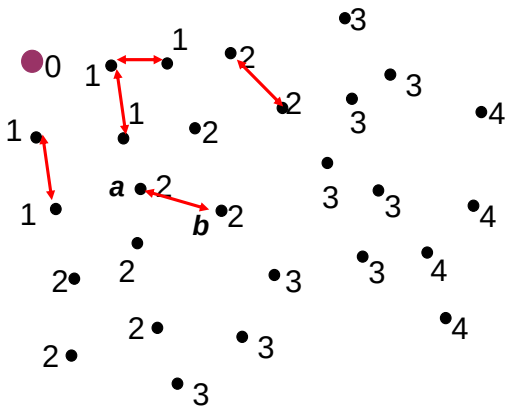


Figure 3.2: Different nodes with different distance to a bootstrap node share the same hop count. In this figure, after hop-counting, many different nodes (represented as small black dots), which have different distance to a bootstrap node (shown as big red dot), share the same hop count. Some examples of this are indicated by arrows.

3.2.3 Connectivity Technique

Connectivity technique is another popular Range-Free approach that needs no special hardware. This technique is described in detail in related research work in Chapter 2. A unit disk graph (UDG) is used to represent connectivity information. Connectivity technique uses the connectivity information to represent the topology of a network. Connectivity information is used to measure the distances between pairs of nodes inside of the network.

3.2.4 Problems with Connectivity-based Technique

Both hop-counting and connectivity techniques use connectivity information to measure the distance inside a network. Similar to the hop-counting technique, the connectivity-based technique also has the problem that many nodes share the same measured distance even if they do not have the same physical distance to other nodes.

3.3 Other Approximations

Since both connectivity and hop-counting techniques are not so accurate for measurement. People are trying to develop some approximations to overcome obstacles to improve the accuracy of measurements.

3.3.1 Kleinrock-Silvester (KS) Formula Approximation

Nagpal et al. demonstrates that the average number of neighboring nodes has an important role in computing the distance based on the connectivity [60]. They computed the distance based on Kleinrock-Silvester (KS) formula [44], which gives the correlation between the hop approximation a_{hop} and the number of neighbors N_i as:

$$a_{hop} = 1 + e^{-|N_i|} - \int_{-1}^{1} e^{-\frac{|N_i|}{\pi}\left(\arccos(t) - t\sqrt{1-t^2}\right)} dt, i \in N_i \tag{3.1}$$

Using this a_{hop}, it is easy to compute the distance d from some source node to node i, if we already known the communication range R:

$$d = a_{hop} * hop_i * R \tag{3.2}$$

But unfortunately, KS approximation only considers the density of a wireless sensor

network. The approximation does not consider the number of hops that a measurement message has traveled to reach the target node, that is also related to hop_{appr} [92].

3.3.2 Elastic Localization (ELA) Approximation

Pascal et al. empirically evaluated the average physical distance from a given sensor node to each of its neighbors or beacons using the number of hops and the number of neighbors. They proposed Elastic Localization Approximation (ELA) [92]. Their idea is to utilize not only the number of neighboring nodes, but also the number of hops that have been traveled. For each value of the number of neighboring nodes, varied from one to thirty, they generated twenty networks of 5000 sensors. In each network, they computed the hop distances from each sensor to every other one. Using values of the hop distance and the number of neighboring nodes, they finally evaluated the average distance. ELA approximation was derived based on the data resulting from afore mentioned simulations:

$$a_{hop} = \left(1 - (\tfrac{2}{3})^{hop_i - 1}\right) \frac{2\arctan\left((\frac{|N_i|}{8})^{\frac{3}{4}} \times \sqrt{3}\right)}{\pi} + \left(\frac{2}{3}\right)^{hop_i}, i \in N_i \tag{3.3}$$

After a_{hop} is known, the following equation can be used to estimate the distance with previously known communication range R:

$$d = a_{hop} * hop_i * R \tag{3.4}$$

Compared to KS approximation, ELA approximation uncovers the relation between the hop approximation and the number of hops traveled. But it still loses much information that could help to improve the accuracy of the measurement. An important observation is that such approximations do not consider the difference in their neighboring nodes' locations. For example, if the neighboring nodes of one node change their locations slightly, but do

not leave that node's neighborhood, the above approximations will not discover such a movement.

3.4 Hop Coordinates

We find that, in reality, there is more information hidden in the neighbors of a node. The hop coordinates technique tries to collect and utilize this additional information, which current hop-counting and connectivity-based techniques do not do.

3.4.1 Definition of Hop Coordinates

Unlike other measurement techniques used in localization, which only count the number of hops or utilize connectivity between nodes, the proposed hop coordinates approximation not only counts the number of hops from some bootstrap node, but also offsets it with the local network information of that node.

Let us still use node a and b in Figure 3.1 as an example to explain the idea behind hop coordinates. From Figure 3.1, we can see that it is true that a node comes with large number of hop, if that node is far away from the bootstrap. In other word, a node that is far away from the bootstrap has neighbor nodes, for which there are more neighbor nodes that have large hop. From Figure 3.1, we can find out that for node a, there are five neighboring nodes with 1 hop, 4 neighboring nodes with 2 hops, while for node b, there is just one neighboring node with 1 hop, five neighboring nodes with 2 hops, and three neighboring nodes with 3 hops. Therefore, by using the hop information that are hiding in the neighboring nodes, we define hop coordinates:

Definition 1: A ***hop coordinates*** is constructed from two parts: *number of hops* and *offset*. The first part is a positive integer that equals the number of hops in a minimum hop route from some bootstrap node to a given node. The second part can be seen as a decimal

fraction generated from local network information, and is defined as:

$$hop_i = \begin{cases} \min_{j \in N_i}(hop_j) + 1, when\,|N_i| \neq 0; \\ +\infty, when\,|N_i| = 0. \end{cases} \tag{3.5}$$

$$\textit{offset}_i = \begin{cases} \frac{\sum\limits_{j \in N_i} (hop_j - (hop_i - 1)) + 1}{2(|N_i|+1)}, when\,|N_i| \neq 0; \\ 0, when\,|N_i| = 0. \end{cases} \tag{3.6}$$

Here, i and j are nodes, hop_i is the minimum number of hops to reach node i counting from some bootstrap node, N_i is a set of nodes that can be reached by node i in a single hop, and $|N_i|$ is the number of nodes in N_i.

We also define bootstrap node in a network is the node with $hop = 0$ and $coor = 0$, and the first node to send out a bootstrap message.

LEMMA 1: For an arbitrary node i, its *offset*$_i < 1$ and $hop_i \geq 0$.

Suppose that N_i is a set of nodes that can be reached by node i in a single hop. If N_i is empty, then *offset*$_i = 0$ and $hop_i > 0$.

If N_i is not empty, then $\textit{offset}_i = \frac{\sum\limits_{j \in N_i} (hop_j - (hop_i - 1)) + 1}{2(|N_i|+1)} = \frac{\sum\limits_{j \in N_i} (hop_j - hop_i) + |N_i| + 1}{2(|N_i|+1)}$. Suppose that node j is an arbitrary node in N_i. Because the hop distance between node i and j is at most 1 hop, $|hop_i - hop_j| \leq 1$, so, we get $-|hop_i - hop_j| \leq hop_i - hop_j \leq |hop_i - hop_j|$, so $\frac{-|N_i|+|N_i|+1}{2(|N_i|+1)} \leq \textit{offset}_i \leq \frac{|N_i|+|N_i|+1}{2(|N_i|+1)}$, but because there is at least one node (assuming node k, k can equal j) in N_i who sends bootstrap message with $hop_k \leq hop_i - 1$ to node i. Thus, we can say that $0 \leq \textit{offset}_i < 1$, for an arbitrary node i.

After hop coordinates is known, we can use the following equation to estimate the distance between two nodes i and j:

$$d.est = (hop_{(i,j)} + \textit{offset}_{(i,j)}) * R \tag{3.7}$$

Here, R is the radio range for a node in the network. Since we assume that the WSNs we studied are homogeneous, each node in a WSN shares same R.

3.4.2 Process to Compute Hop Coordinates for Hop-Counting

In order to simplify communication, we assume that WSNs are symmetric, which means that suppose node i and j are neighbors, if node i can reach node j, then node j can reach i. These assumptions are reasonable in simulation, since our technique is simulated above the 802.15.4 MAC layer [106] in NS-2 [57], which guarantees the communication to be symmetric. In NS-2, every message transmitted in the 802.15.4 MAC layer from node i to node j will be followed up by an *ACK* message from node j to node i. We will discuss the problem if an ACK message is dropped in the discussion section in chapter 7.

Besides the type of ACK message, there are two types of messages used in a running hop coordinates protocol — bootstrap message and neighboring message. Bootstrap message is used to count the number of hops for each node. Neighboring message is used for a node to retrieve hop coordinates from its neighboring nodes. Both types of messages share the same message structure in Table 3.2:

Size (byte)	2	2	2
Content	Origin nodeID	Destination nodeID	Previous nodeID
Size (byte)	2	1	22
Content	Next nodeID	Message type	Message data

Table 3.1: The message structure used in hop coordinates

The descriptions for the fields in this message structure are as follows:

Origin nodeID: the ID of the node that originally sends out this message.

Destination nodeID: the ID of the destination node that the message is to be sent to.

Previous nodeID: the ID of the node that the message was received from.

Next nodeID: the ID of the node that the message is to be sent to.

Content	Size (byte)
Origin nodeID	2
Destination nodeID	2
Previous nodeID	2
Next nodeID	2
Message type	1
Message data	22

Table 3.2: The message structure used in hop coordinates

Message type: a byte that indicates the type of this message.

Message data: a data field in the message. In a bootstrap message, we assign this memory to hop variable. In a neighboring message, we assign this memory to hop and *offset*.

To compute hop coordinates in a WSN, we first appoint an arbitrary node as the bootstrap node. The bootstrap node generates a bootstrap message, which includes variable *hop=0*. This message is flooded across the whole network, allowing each other node to count a hop distance from itself to the bootstrap node. After a small constant TIMEOUT, all nodes can calculate their own hop coordinates. The detailed processes for both the bootstrap node and other nodes are shown in the following:

1. In bootstrap node:

Assigned bootstrap node, which $hop = 0$ and *offset*$= 0$, initializes a bootstrap message with *hop=0* to flood the network; after that, the bootstrap node drops any message originated by itself.

2. In other nodes in the WSN:

At first, we initialize $hop =$*MAX* (a constant maximal hop number) and *offset*$= 0$in each node except bootstrap node. The following algorithm is run in every node, say a, which is not the bootstrap node, as Procedure 2.

Procedure 2 Hop coordinates in node a

for bootstrap message (hop_b) from any $b \in N_a$ and not TIMEOUT **do**
  if $hop_b + 1 < hop_a$ **then**
    $hop_a = hop_b + 1$
    transmit (bootstrap message (hop_a))
  else
    drop (bootstrap message (hop_b))
  end if
end for
request hop with neighboring message from any $b \in N_a$
if $|N_a| == 0$ **then**
  $\mathit{offset}_a = 0$
else
  $\mathit{offset}_a = \frac{\sum_{b \in N_a}(hop_b - (hop_a - 1)) + 1}{2(|N_a| + 1)}$
end if
return hop_a and offset_a

3.4.3 Process to Compute Hop Coordinates for Connectivity-Based Technique

In the connectivity-based technique, a pre-assigned bootstrap node is not necessary, since every node does bootstrap function for its local area, while there are multiple nodes that can function as bootstrap node. Therefore, we will not only generate a particular hop coordinate for a particular bootstrap node.

Here we still use the same message structure as shown in Table 3.2. The idea to implement hop coordinates in the connectivity-based technique is to allow each node to be a bootstrap node. Not as in Section 3.4.2, there is only one bootstrap in a network, here each node works both as a bootstrap node to create bootstrap message and a normal node to forward bootstrap messages from other nodes. So, each node can receive multiple bootstrap messages from different nodes. To deal with multiple bootstrap messages, each node maintains a table (called op-coordinates table) to store hop received from its neighboring nodes. Each entry in this table has three columns as shown in hop-coordinates table (Table

3.3): index column is *nodeID* and data columns are hop and $hop.coor$. At first, we initialize every hop in the table to equal *MAX*.

index	data	bit	data	bit
nodeID	hop	*hop-ready*	$hop.coor$	*ready*

Table 3.3: The table structure for hop-coordinates table in connectivity-based hop-coordinates procedures

Here, suppose that this table (Table 3.3) belongs to node i, *nodeID*, which is a unique ID for each node in a network, represents the ID for one of node i's neighboring nodes, hop is the hop count originated by node *nodeID* to node i, and $hop.coor$ is the hop coordinates originated by node *nodeID* to node i.

The main procedure is shown in Procedure 3. For simplification, here we only consider one node, say node i. At first node i initializes its hop-coordinates table by setting hop as maximal hop number *MAX*, *hop-ready* and *ready* as zero. Then node i broadcasts out a bootstrap message that includes its ID, a hop count variable $hop_i = 0$ and a constant *HOPMAX*. The neighboring nodes (say k) of node i receives that message will forward that message by adding hop_i with one hop unless *HOPMAX*$> hop_i + 1$ (see Procedure 4).

After a time TIMEOUT, for each node j in N_i, node i sends out a neighboring message to retrieve the hop counts from its neighboring nodes, which is originated by node j, then computes a hop coordinates and stores it into the corresponding entry in the hop-coordinates table in node i. If node i receives a bootstrap message from some other node, Procedure 4 is called in node i to process this message. Procedure 5 deals with the inbound neighboring message.

Not like hop coordinates procedures in hop counting, where there is only one bootstrap node, in connectivity-based hop coordinates, each node works both as a bootstrap and a normal node at the same time. So, after the hop coordinates procedure, each node will have a hop coordinates table, in which each hop coordinates distance to each of its neighboring

Procedure 3 Connectivity-based hop coordinates in node i

hop-ready = 0
origin a bootstrap message ($hop_i = 0$) to any node $j \in N_i$
wait until other nodes receive bootstrap message or TIMEOUT
for any $j \in N_i$ and not TIMEOUT **do**
    retrieve all $hop_{(j,k)}$ originated by node j from each node $k \in N_i$ with neighboring messages
    if $|N_i| == 0$ **then**
        delete the entry indexed by j from the hop-coordinates table
    else
        $\mathit{offset}_{(j,i)} = \frac{\sum_{k \in N_i} (hop_{(j,i)} - (hop_{(j,k)} - 1)) + 1}{2(|N_i| + 1)}$
    end if
    update $hop.coor$ column in the entry indexed by j in the hop-coordinates table
    update *ready* column in the entry indexed by j as non-zero in the hop-coordinates table
end for

Procedure 4 Bootstrap message receiver procedure in node i

INPUT: bootstrap message ($hop_{(j,k)}$) from node k originated by node $j \in N_i$
if *HOPMAX*$< hop_{(j,k)} + 1$ **then**
    drop bootstrap message
    exit
end if
if $hop_{(j,k)} + 1 > hop_{(j,i)}$, which is originated by node j **then**
    drop (message ($hop_{(j,k)}$))
    exit
end if
$hop_{(j,i)} = hop_{(j,k)} + 1$
update the hop column in the entry for node j in the hop coordinates table in node i
forward bootstrap message ($hop_{(j,k)} + 1$)
hop-ready = 1 in the entry indexed by j

Procedure 5 Neighboring message receiver procedure in node i

INPUT: neighboring message to request $hop_{j,i}$, which is originated by node j, from node $k \in N_i$
wait until (*hop-ready* =1)
Send back hop-coordinates table[index = j].hop

nodes is stored. This table can be used in location computation, which is to be talked about in detail in Chapter 4, 5 and 6.

3.5 Simulation Result

We implemented our measurement algorithm as a routing agent and our bootstrap node program as a protocol agent in NS-2 version 2.29 [57] with 802.15.4 MAC layer [106] and CMU wireless [28] extensions. The configuration used for NS-2 is RF range = 15 meters, propagation = TwoRayGround, antenna = OmniAntenna. We simulated different measurement techniques and compared them in the accuracy of measuring distance on different network placements with varying number of nodes and network densities.

3.5.1 Metrics

In order to compare the accuracy of different approximations, we define the error of accuracy for the measurement based on different approximations for each node. Suppose that the physical location of a node i is $c_i = (x_i, y_i)$ in 2D, and that the physical location of a node j is $c_j = (x_j, y_j)$, so the distance between node i and node j is $d_{(i,j)} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. Suppose the distance measured by an approximation is $d.est_{(i,j)}$. We can measure the error by comparing the measured distance to the physical distance.

$$error = \left| d.est_{(i,j)} - d_{(i,j)} \right| / d_{(i,j)} \times 100\% \tag{3.8}$$

We also define the Network Density (*ND*) as the average number of nodes in one hop transmission range.

3.5.2 Simulation Process

The ideas of hop-coordinates for hop-counting and hop-coordinates for the connectivity-based technique are quite similar to each other. In the hop-counting technique a single bootstrap node exists, whereas each node works as a bootstrap node for itself in connectivity-based technique. We implement the hop coordinates algorithm as a routing agent in NS-2 version 2.29 [57]. The configuration used for NS-2 is RF range = 15 meters (which equals radio range R used in this dissertation), propagation = TwoRayGround, antenna = OmniAntenna. The propagation range for a node is controlled to stop forwarding the hop-coordinates message by the number of hops k from 1 to 43 (43 is the maximum hop distance for the purpose of our simulation).

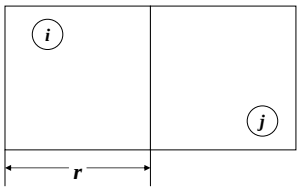


Figure 3.3: A node placement in our experiment. Here, the circled i means node i. We suppose the deployment area for a WSN is a square. We separate the whole square into small squares with their width equal to r. Each node is placed somewhere in its square that is placed uniformly randomized in any place inside this small square.

We use uniform placement — n nodes are placed on a grid with a uniform randomized placement error r, where r is the width of a small square inside the grid. An example node we will be placing is shown in Figure 3.3. We constructed a total of 60 placements with number of nodes n = 36, 100, 225, 400, 625, 900, 1600, 2025, and 2500, and with r =2, 4, 5, 8, 10, and 12 meters, respectively. The reason we use uniform placement with *100%r* error is that such placement usually includes both node holes and islands within a single placement. One example of a 400-node network is illustrated in Figure 3.4. For hop-counting based hop coordinates, we select a node, which is sitting in one corner of the square area where the network deploys, as the bootstrap, so that it can reach the longest

hop distance.

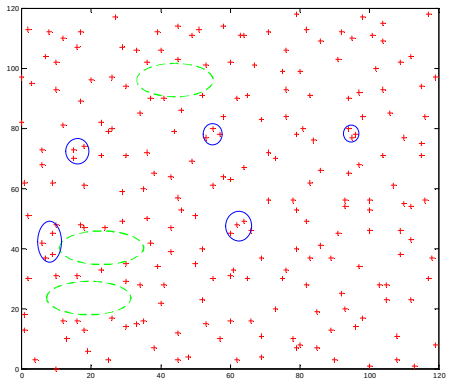


Figure 3.4: A typical placement for simulation. The placement is constructed with $n = 400$ and $r = 4$m in a square area 120mX120m. Here "+" represents a node. The dashed ovals (also in green) are holes, which are the large areas where there are no nodes, and the small circles (also in blue) are islands, where are the large areas where there are many nodes crowded together.

The simulator is then run on these placements to get a hop coordinate for each node in each placement.

At the same time, we use the same placement as well as the same bootstrap node to generate the experimental data for hop-counting technique. The experimental data for ELA and KS are generated by importing the number of neighboring nodes and the hop distance for each node in each placement into their equations.

Now, for each placement, we have four data files, one each for hop-coordinates, hop-counting, ELA and KS. After that, we can compute $d.est$ for each node to the bootstrap based on four methods to the bootstrap. And we can measure d for each node to the bootstrap node. By using the Equation 3.8, we can calculate the accuracy of measurement for each node.

3.5.3 Simulation Results

We obtained results of the accuracy of the measurement by ELA, KS, hop-counting and hop-coordinates techniques using networks with n = 36, 100, 225, 400, 625, 900, 1225, 1600, 2025, and 2500 nodes, and r = 2, 4, 6, 8, 10, and 12 meters (which corresponds to Network Density (ND) = 144, 38, 18, 10, 6, and 4, respectively), respectively. They are shown in Figure 3.5 to Figure 3.10. In the figures, we use a diamond is used to represent the result of ELA, a square to represent the result of KS, a triangle to represent the result of hop-coordinates, and an "x" to represent the result of hop-counting. Each data point represents ten repetitions.

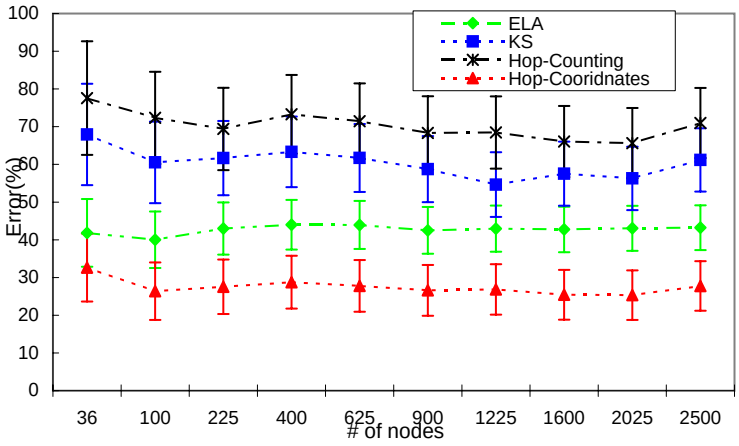


Figure 3.5: Comparison of the accuracy of different approximations, when $r = 2$m. Here: X-axis is number of nodes and Y-axis is the average error in percentage. Each error bar represents one standard deviation.

From Figure 3.5 we can see that when $r = 2$m, the accuracy in measurement with hop coordinates is higher than it is with other approximations, except when $n \leq 100$. This is because in a network of $r \leq 2$ and $n \leq 100$, for instance when $n = 36$, the longest hop distance in a network with $r = 2m$ is only 2 from our observation. When a node has few diversity in the hop count of its neighboring nodes, our hop coordinates technique has worse performance in accuracy. However, other methods do not perform well either from

Figure 3.5.

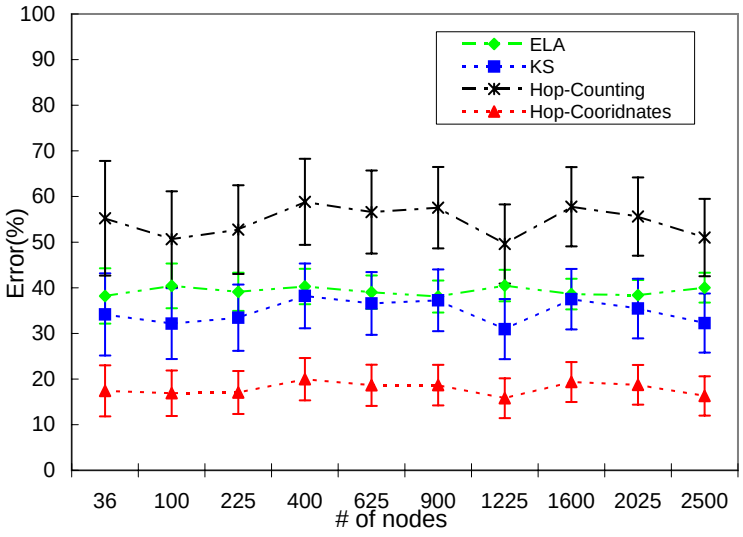


Figure 3.6: Comparison of the accuracy of different approximations, when $r = 4$m.

From Figure 3.6 and Figure 3.7, we can see that when $r = 4$ and $r = 6$ meters, hop-coordinates still outperforms other methods with respect to accuracy. Comparing with Figure 3.5, when $r = 4$m and $r = 6$m, the standard deviation of the performance for hop coordinates in accuracy is better than in the case when $r = 2$m, which means that in these two cases ($r = 4$m and $r = 6$m) the performance is more stable than when in $r = 2$m.

In Figure 3.8, we can see that hop-coordinates outperforms hop-counting as well as ELA methods. From the results, we observe that KS performs so well is that the idea of the KS equation comes from fitting the curve of some real experiment [60] and hence the KS equation is optimized for the case close to this particular case (when $r = 8$m).

Our technique does not require each node to connect to more than a certain number of neighboring nodes. But when $r \geq 10$m, the average number of neighboring nodes is down to ND = 6 (when $r = 10$m) or ND = 4 (when $r = 12$m). Figure 3.9 and Figure 3.10 show that in such a situation the hop coordinate technique has closer performance to other methods than when $r < 10$m. From Figure 3.10, we can see when the ND is very small,

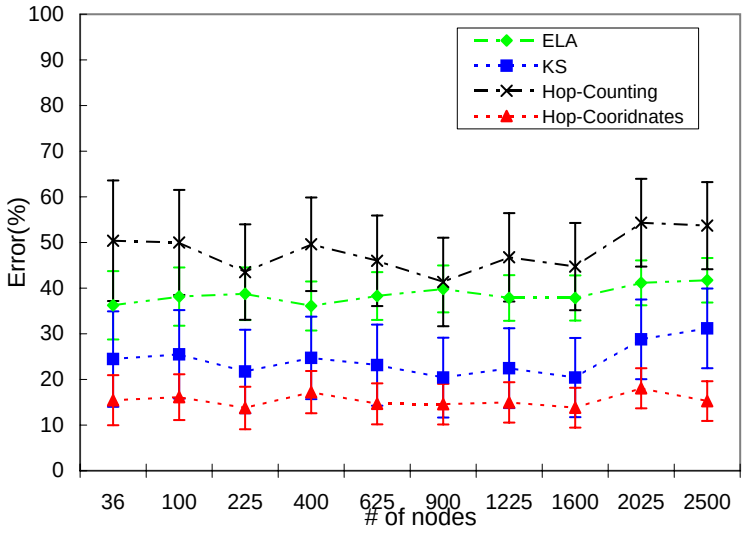


Figure 3.7: Comparison of the accuracy of different approximations, when $r = 6$m.

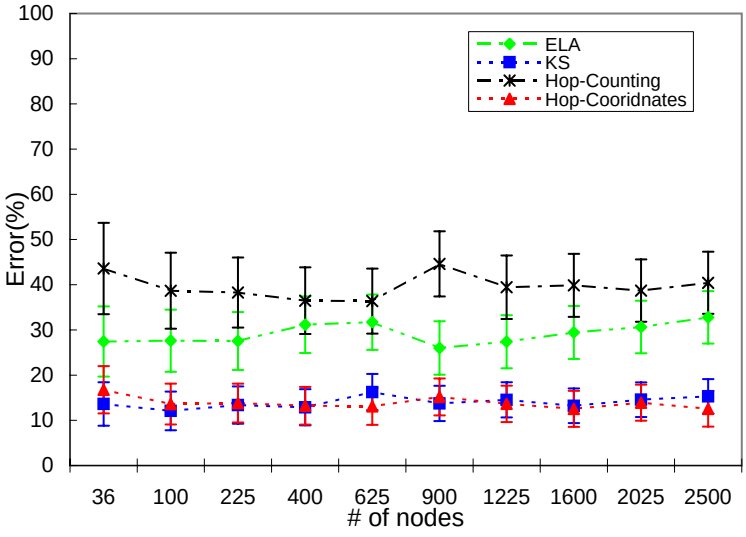


Figure 3.8: Comparison of the accuracy of different approximations, when $r = 8$m.

the performance of all methods is largely affected by the network structure.

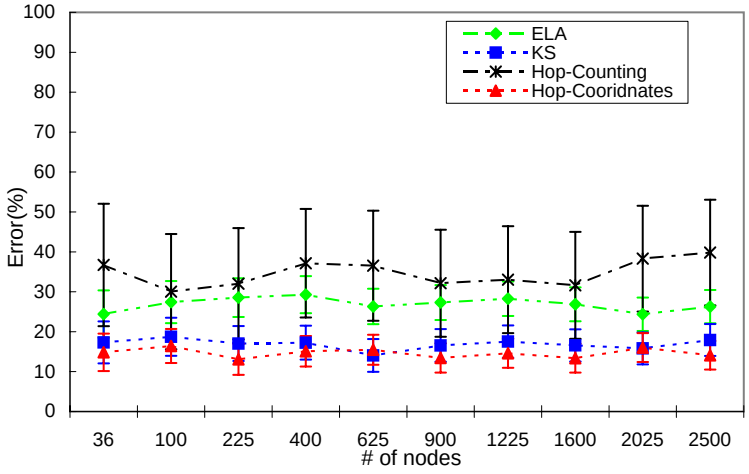


Figure 3.9: Comparison of the accuracy of different approximations, when $r = 10$m.

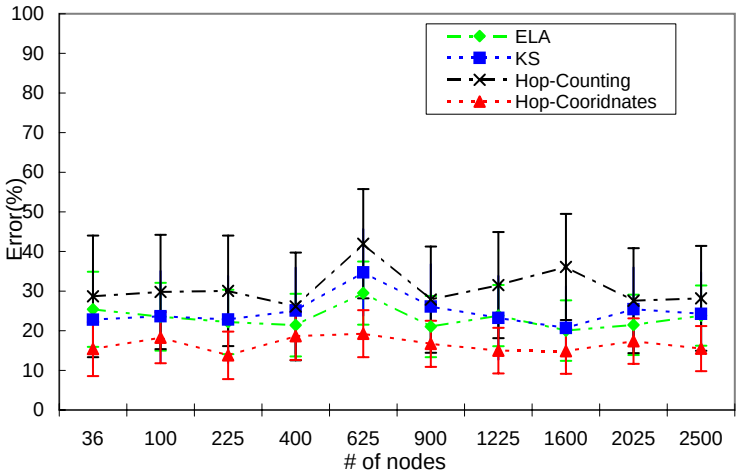


Figure 3.10: Comparison of the accuracy of different approximations, when $r = 12$m.

In Figures 3.5 to 3.10 we can see the improvements in accuracy when the hop coordinates are stable in the testbed network. This is true from a small scale of 36 nodes to the largest scale of 2500 nodes, if the diameter of a network is not less than one hop distance.

The overall average accuracy is calculated based on Figures 3.5 to Figure 3.10, corresponding to r=2, 4, 6, 8, 10, 12 meters. The results are shown in Figure 3.11. From this

figure, we can see that compared with ELA, KS, and hop-counting techniques, the accuracy with our technique increases about 47%, 39% and 62% respectively when $2 \leq r < 10$m. Only the KS method is close to our method in performance when ND =10 (or $r = 8$m).

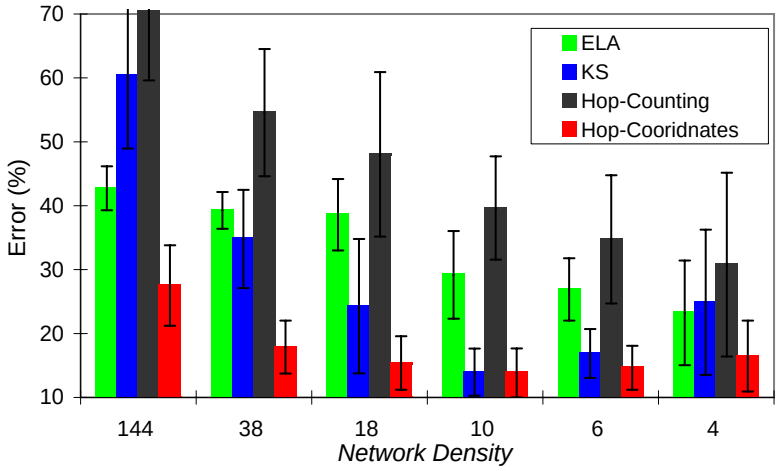


Figure 3.11: Comparison of overall average accuracy of ELA, KS, hop-counting and hop-coordinates. Here, each error bar represents one standard deviation.

Another problem we should consider is whether our technique is stable with respect to the hop distance. The hop distance is defined as the distance from one node to another node here. Based on the data we got from the accuracy experiment, we resorted them based on the number of hops. We collected nodes with different hop distance to the particular bootstrap, and then estimate their distance to that bootstrap with ELA, KS, hop counting and hop coordinates methods, respectively. The results are shown in Figure 3.12. We can see that except when the number of hops equals one, the error of our method is higher and the performance of our method is very good even when the hop distance is long.

3.6 Summary and Discussion

Here, we focused on how to improve the accuracy of measurement in localization and found that problems exist in current hop-counting and connectivity-based measurement

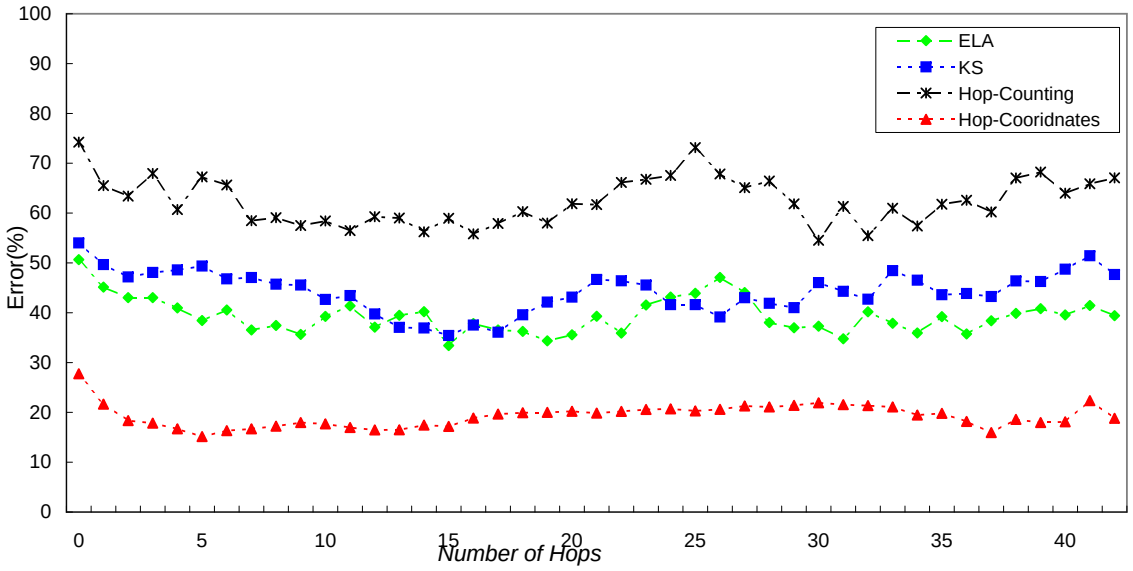


Figure 3.12: Comparison of the Accuracy of ELA, KS, hop counting and hop-coordinates with varying number of hops. Here: X-axis: hop distance in term of number of hops, Y-axis: the average error in percentage.

techniques. We presented a new localization technique called hop coordinates, which improves the accuracy of the measurement while using only connectivity information.

In our simulations, the proposed hop coordinates technique improves the accuracy of measurements generated by ELA, KS and hop-counting by 47%, 39% and 62% (by summarizing data in Tabel 3.11), respectively, when compared to the results when our technique is not applied. Our technique works well in networks with varying numbers of nodes. Current distributed localization algorithms do request that the performance of measurements be good even when the hop distance is short. We observed from our simulations that our technique still performs well when the hop distance is short.

Chapter 4

Anchor-Free Localization in $(n, 0)$ WSNs

As first step for localization in mixed (n, m) WSNs, we talk about how to do localization without priori in $(n, 0)$ WSNs in this chapter. Localization in $(n, 0)$ WSNs, in other words, static WSNs, in which all the nodes are assumed to keep sitting in a permanent position, is a special case for localization in mixed WSNs. In this chapter, we will introduce a new distributed localization algorithm called Geographic Distributed Localization (GDL) that is based on hop coordinates.

4.1 Introduction

In this chapter, we introduce a new localization algorithm—Geographic Distributed Localization (GDL), that generates a more accurate location for each node based only on hop-counting and without anchor nodes with the help of hop coordinates (Section 3). Although the idea of predicting node position from connectivity information in WSNs is not new, our algorithm improves on other localization algorithms of this type in several key ways.

Our algorithm generates hop coordinates for each node by combining local network

connection information with hop numbers generated without anchor nodes. We show how a distributed version of this algorithm can be made with constant costs in terms of the amount of memory, computation and communication overhead required per node and $O(n)$ for an overall n-node network. Based on extensive simulations in NS-2, our localization algorithm shows accuracy improvements of about 45% and 36%, respectively, in effective range, compared with the localization algorithms MDS-MAP [83] and MDS-MAP(P) [82], which outperform the other existing localization algorithms of this type under these conditions [82, 83]. It also has a lower communication cost ($O(n)$ versus $O(n \log n)$) than MDS-MAP(P), and has a constant memory cost per node.

4.2 Geographic Distributed Localization Algorithm (GDL)

This section presents a new distributed localization algorithm called GDL that is based on hop-coordinates technique.

4.2.1 Description of the Algorithm

Our GDL is a distributed localization algorithm, which relies on the measurement technique talked in Chapter 3. Though, here GDL runs on the top of hop coordinates, GDL can be compatible to future measurement techniques. Unlike other localization algorithms, our approach not only counts the number of hops from some bootstrap node but also offsets this count with local network information specific to a given node.

We also assume that:

- The communications in WSNs are symmetric. Which is reasonable, since our algorithm runs on network layer, and the packages are sent to MAC layer, which can take care of whether the package received or not.

- Each node has a unique ID, which is a pre-defined number that can uniquely identify that node. Here we use digits to represents IDs. Providing a unique ID for each node is generally easy to achieve in practice, such as when nodes are pre-initialized uniquely by the manufacturer.

Our GDL algorithm can be summarized in five steps:

Step 1: Select two proper bootstrap nodes from b bootstrap node candidates

Before we deploy a WSN, we assign b (which is a constant, in practice $b = 8$ for all n from 36 to 2500 placements) bootstrap node candidates from total n nodes in the network. We mix these b bootstrap node candidates into the total n nodes randomly. Then, we deploy these n nodes as a network in an area, to simplification, a square area.

Step 1.1:

Every bootstrap node candidate floods a message out to count the hop distance from itself to every other bootstrap node candidate.

Step 1.2:

After Step 1.1, each bootstrap node candidate has a $b \times 1$ pair-distance vector for the b candidates, each bootstrap node candidate then broadcasts this vector to other candidates. Right now each bootstrap node has a $b \times b$ pair-distance vector for the b candidate. Then each candidate compares the distance between each pair of candidates, letting the pair of candidates with the longest distance "win". In case there are several pair candidates with the same longest distance, then we resolve ties by selecting the pair of candidates in order of their *ID* (starting with the smallest) until we have selected one linked pair as the winners. This results in the deterministic selection of two bootstrap nodes, specified as X and Y

(here we can let the node with the smaller *ID* be X and the other one Y), which will help us to generate the hop coordinates and separate the whole network into geographically distributed local areas. In Step 1.1 and Step 1.2, the communication cost for each node is at most $O(b^2)$, the communication cost for the whole network is at most $O(b^2 * n)$ (if we assume all communications are using broadcasting).

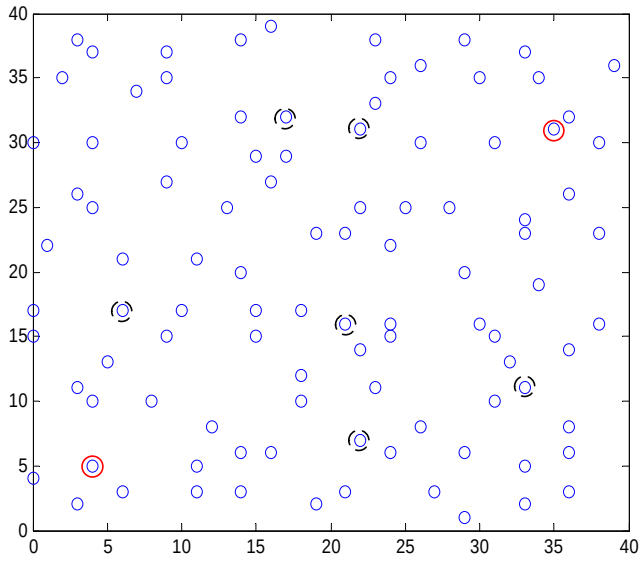


Figure 4.1: Bootstrap node selection in a 10*10 WSN. Here, two circled nodes were selected as bootstrap nodes X, Y from 8 candidates, which are dash circled. X, Y axes are in meter.

In this step we incur a computational cost $O(b^2)$, a memory cost $O(b^2)$, and an $O(b^2)$ communication cost, per node, and an $O(b^2n)$ communication cost for the whole networks (if, in the worst case, we use flooding to communicate). Because there are a fixed number b of bootstrap nodes, each has a constant cost in memory, computation, and communication. Figure 4.1 shows how in a WSN with 100 nodes we ultimately select the two nodes with dashed circles as bootstrap nodes.

Step 2: Compute hop coordinates

The detail of how to compute hop coordinates has been introduced in Chapter 3, so, here we describe it briefly.

Step 2.1:

Given two bootstrap nodes X and Y assigned in Step 1, initialize with one message from each with variables hop_X and hop_Y, respectively, to flood the network.

Step 2.2:

Every other node records its number of hops from bootstrap nodes X and Y, and computes the integer part of its hop coordinate.

The detailed processes in each bootstrap node and in other nodes are as follows:

(i) In bootstrap node: A bootstrap node (X or Y) creates a message with ($i = X$ or Y) to flood the network. After that, the bootstrap node will drop any message that was originated by itself.

(ii) In all other nodes in the WSN: the Procedure 2, introduced in Chapter 4, runs in any node a that is not a bootstrap node to compute the hop coordinate for the node, which means the hop coordinates distance to the bootstrap node.

Step 3: Generate local center nodes

In this step, the network first separates the whole network area into several geographically local areas based on hop coordinates generated in Step 2, and then select a local center node for each local area. After that, it will be used to take care of computation of the local map in that area in Step 4 and Step 5.

Step 3.1:

Elect center node for local area.

With hop coordinates $hop.coor_X = 1/2 * i$ (i ranges from 1 to N) and $hop.coor_Y = 1/2 * j$ (j from 1 to N) as virtual boundaries, we separate the whole WSN into many local areas. We then select a node in each local area as a center node for that area. Ideally, a *center position* should be $hop.coor_X = 1/2 * i + 1/4$ and $hop.coor_Y = 1/2 * j + 1/4$ for a local area bounded by $1/2 * i < hop.coor_X \leq 1/2 * i + 1/2$ and by $1/2 * j < hop.coor_Y \leq 1/2 * j + 1/2$.

In order to select a center node for each local area, we employ a simple voting mechanism that selects a node as near as possible to the ideal center position c (in the following text, we refer to this position as the "ideal local center"). The basic idea behind this simple voting mechanism is to let node delay more time if it is far away from c. We use the distance from this node to c as the parameter for a delay function f.

To vote, each node a in that area delays a period of time:

$$t = f(\sqrt{(hop.coor_{(a,X)} - hop.coor_{(c,X)})^2 + (hop.coor_{(a,Y)} - hop.coor_{(c,Y)})^2}) \qquad (4.1)$$

For function $f(x)$, the simplest expression we can use is Cx, where C is a constant. In the period of time until t, the node will listen to other nodes in its area, and if no messages are received, then the node will send a message to vote itself as a center node of that local area. The whole procedure including a collision detection provision is shown in Procedure 6.

In our simulation, this simple voting algorithm works well, even in some exceptions like the one shown in Figure 4.2, where the voting mechanism produces more than one center node for a single local area. In the local area circled in red, three local center nodes are produced because the distances between some nodes are more than one hop (In Figure

Procedure 6 Vote for Center node in node a

1: INPUT: $hop.coor_{(a,X)}$, $hop.coor_{(a,Y)}$, ID_a, $hop.coor_{(c,X)}$, $hop.coor_{(c,Y)}$, here c is the local ideal center.
2: delay($f(\sqrt{(hop.coor_{(a,X)} - hop.coor_{(c,X)})^2 + (hop.coor_{(a,Y)} - hop.coor_{(c,Y)})^2})$)
3: SEND:
4: **if** Not $receive(msg(center_a))$ from nodes in local area **then**
5: $center_a = self$
6: **while** $send(msg(center_a))$ == collision **do**
7: delay($f(\sqrt{(hop.coor_{(a,X)} - hop.coor_{(c,X)})^2 + (hop.coor_{(a,Y)} - hop.coor_{(c,Y)})^2} + random(ID_a)$)
8: goto SEND;
9: **end while**
10: **end if**
11: return $center_a$

4.2, since the middle circled node is closer to the ideal center node c position, then, it wins the vote in the area bordered by the two carves and the two strait lines. But the nodes in the areas except the above area, which already generated a winner, will continue to vote, so beside the middle circled center node, two other center nodes, which are circled, are generated for that area.).

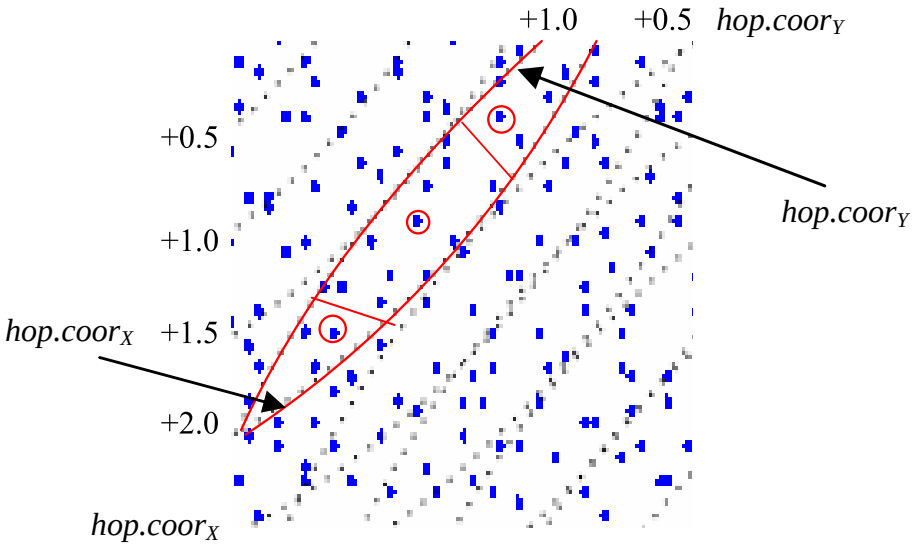


Figure 4.2: An example of center node generation in a 2500-node WSN. The nodes with circles are selected as center nodes for one local area, whose diameter is > 1 hop.

Step 3.2:

After the generation of at least one center node for each local area each center node will send a request to its neighbor nodes that are within k hops to send back their hop coordinate from bootstrap node X or Y.

The total cost for Step 3 is as follows: computational cost of $O(1)$, communication cost of $O(|N_a|)$, memory cost of $O(|N_a|)$ for center nodes and $O(1)$ for nodes that are not center nodes, and $O(n)$ for the whole network.

Step 4: Apply MDS algorithm to calculate the local map in each center node

After we selected center nodes for each local area, in Step 4, each center node creates a pair distance matrix for all nodes in the local area, and then applies Multidimensional Scaling (MDS) algorithm (The basic concept of MDS is introduced in the appendix of this chapter (in Section 4.5)) to the distance matrix to generate a local map for each center node.

Step 4.1:

Each center node sends out a message to request the hop coordinates measurement from every nodes in its local area, and then computes shortest paths between all pairs of nodes one $1(k)$ hop(s) to that node, using Dijkstra's algorithm or other similar algorithms.

Step 4.2:

We then apply classic MDS to the ($|N_a|$+*1*)×($|N_a|$+*1*) shortest path matrix (here $|N_a|$ is the number of nodes that can be reached by center node a in k hops) and retain the first two largest eigenvalues and eigenvectors to construct a 2-D local map $F_(a)$, the coordinates of the nodes in this local map are stored in an estimated coordinate matrix $F_a = [..., f_j, ...]$ in node a, where f_j is a estimated coordinate for node j in the neighboring area of node a

($j \in N_a \cup a$), the coordinate system of this local map (represented as $F_(a)$) are normalized only for this local area by MDS.

Because classic MDS requests $O(n^3)$, here n is the number of items in the matrix inputted into MDS. The total cost for this step is a computational cost of $O(|N_a|^3 n)$ and a memory cost of $O(|N_a|^2)$ per center node, with no communication cost in this step.

Step 5: Calculate transformation matrix and transform local map into a portion of a global map

After Step 4, each center node has a local map for its local area with its own coordinates system, so, there are many local maps with their own coordinate system in a network. To merge all local maps into a global map, we need to compute a transformation matrix T for each local map to transform the local map into a portion of the global map.

The idea to do this job is as follows: First, randomly assign a center node and start with its local map; then we choose a neighbor center node, whose local map shares the most nodes with the local map that current center node has, to generate a transformation matrix for its neighbor center node's local map, then continue this procedure until there is no center node, whose local map was not transformed.

Step 5.1:

We next find the set of neighboring center node for each center node a.

For this process, we need to randomly assign a node to bootstrap the process. In practice, node Y will be used as the bootstrap, which we will refer to as node a, to transform the location for a center node of a given area by running Procedure 7 and Procedure 4.3(which will be talked in Step 5.2). Except bootstrap node Y, other center nodes will receive transformation matrix T with Procedure 9.

Procedure 7 Find neighboring center nodes for node a

1: Send out an ASK message to ask for possible neighbor center nodes within k hops of node a to provide their neighbor node sets
2: Receive the reply messages from neighbor centers, which include their neighbor node sets
3: Compare the neighbor node set with N_a, put the neighbor center node with a local map, which shares the most nodes with N_a, into neighbor center set $C_a \subseteq N_a$
4: Return neighbor center set C_a

Step 5.2:

After we find the set of neighboring center nodes C_a, we compute a transformation matrix for each node b in the neighbor center set C_a of a center node a to merge the local map in center node b with the local map of center node a together, though both of the local maps are still stored in node a and b separately. The idea to merge these two local maps together is to compute a transformation matrix for one map based on the coordinates of the common nodes that the two nodes share. A linear transformation matrix T with minimizing discrepancy errors is computed to transform the coordinates of the common nodes in one local map to those in the other map. The detail is as follows:

Suppose that there are two sets of neighbor nodes, N_a and N_b, that within k hops of the center nodes a and b, respectively. Their intersection is $I = N_a \cap N_b$, and we use matrix

$$I_a = \left\{\ldots, (x_i, y_i)', \ldots\right\}' \tag{4.2}$$

(Here (x_i, y_i) are the coordinates of a node $i \in N_a$ in the local map F_a generated by node a) to represent the coordinates of nodes in the I generated by node a, $I_a \subseteq F_a$.. And similarly for node b

$$I_b = \left\{\ldots, (\grave{x}_i, \grave{y}_i)', \ldots\right\}' \tag{4.3}$$

Here $(\grave{x}_i, \grave{y}_i)$ are the coordinates of a node $i \in N_b$ in the local map F_b generated by node b) to represent the coordinates of nodes in the I generated by node a, $I_b \subseteq F_b$.

We can then compute a transformation matrix T_b (which is a $2X2$ matrix) such that it minimizes

$$\sum \sqrt{((x_i - \check{x_i})^2 + (y_i - \check{y_i})^2)} \tag{4.4}$$

for all $i \in I$, where

$$I_a = \left\{ \ldots, (x_i, y_i)', \ldots \right\}' \tag{4.5}$$

and

$$I_b \times T = \left\{ \ldots, (\check{x_i}, \check{y_i})', \ldots \right\}' \tag{4.6}$$

The procedure for a center node to generate a transformation matrix for each of its neighboring nodes is given in Procedure 8.

Procedure 8 Compute transformation matrix T for C_a in node a

1: **for** each node $b \in C_a$ **do**
2: request N_b from node b
3: $I = N_a \cap N_b$
4: $I_a = ..., (x_i, y_i), , , .,$ such that $i \in I$ and $(x_i, y_i) \in F_a$
5: retrieve I_b from node b
6: compute transformation matrix T_b such that $\sum \sqrt{((x_i - \check{x_i})^2 + (y_i - \check{y_i})^2)}$ (for all $i \in I$) is minimized, here $(x_i, y_i) \in I_a, (\check{x_i}, \check{y_i}) \in I_b \times T_b$.
7: send matrix T_b to node b
8: **end for**
9: set *self* as "transformed"

If a center node receives a transformation matrix T, then this center node first checks whether it is already transformed or not; if so, this center node will drop the message, and if not, it will apply procedure 9. This procedure will allow the center node, which receives a transformation matrix, to compute a transformation matrix for each of its $|N_a|$ neighboring nodes, and then send it to its neighbor nodes.

Procedure 9 Receive transformation matrix T in node a

1: INPUT matrix T from neighbor center node
2: **if** node a has been transformed **then**
3: drop(message(T))
4: Exit;
5: **end if**
6: set *self* as "transformed"
7: transform the local map F_a with T into a portion of global map H {a portion of H = $F_a \times T$}
8: send back corresponding coordinates in the local map to the neighbor nodes in N_a
9: call procedure 7 to generate C_a
10: **for** each node $b \in C_a$ **do**
11: request N_b from node b
12: $I = N_a \cap N_b$
13: $I_a = ..., (x_i, y_i), , , .,$ such that $i \in I$ and $(x_i, y_i) \in F_a \times T$
14: retrieve I_b from node b
15: compute transformation matrix T_b such that $\sum \sqrt{((x_i - \check{x_i})^2 + (y_i - \check{y_i})^2)}$ (for all $i \in I$) is minimized, here $(x_i, y_i) \in I_a, (\check{x_i}, \check{y_i}) \in I_b \times T_b$.
16: send matrix T_b to node b
17: **end for**

Total cost for this step: computational cost of $O(|N_a|)$ and memory cost of $O(|N_a|)$ for center nodes, and communication cost of $O(1)$ and memory cost $O(1)$ per node for nodes that are not center nodes.

4.3 Simulation Results

4.3.1 Simulation Configuration

We use the same simulation configuration as in previous chapter, to keep the condition as similar as possible.

We use the MDS-MAP series of algorithms [82, 83] for comparing our results. We ran our localization algorithm in NS-2 and used MDS-MAP series algorithms running in Matlab as in [82, 83] under the same placements as mentioned before. We did not include

the MDS-MAP algorithms' refine steps introduced in [82, 83], but note that similar steps can be added to both our algorithm and the MDS-MAP/MDS-MAP(P) algorithms. Since the global map H generated by GDL includes relative coordinates, we need convert these relative coordinates into physical coordinates to compare with corresponding node's real coordinates. Given three real coordinates for three fixed nodes sitting in three corners, we transform these relative maps to a physical map. We did this in both in this Chapter and the following chapters.

4.3.2 Simulation Result

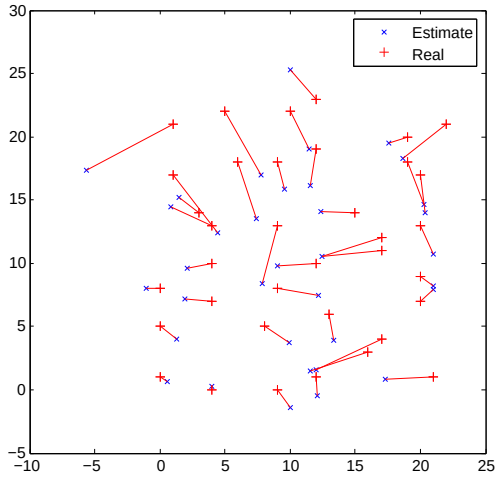


Figure 4.3: Result of GDL in a 36-node network

With the same placement of 36 nodes with r randomized placement error, Figure 4.4 shows the results from our algorithm and MDS-MAP series algorithm. From Figure 4.4, we can see that the comparison MDS-MAP algorithm completes with an error average of E = 1.05 r, while our algorithm (Figure 4.3) completes with an average error of $E = 0.74$ r — about a 34% accuracy improvement.

In order to evaluate the accuracy of our algorithm in different size and different density WSNs, we ran our algorithm and compared the MDS-MAP series algorithms in the 60

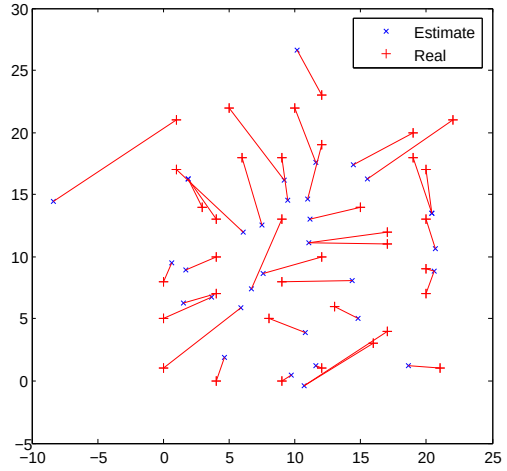


Figure 4.4: Result of comparing MDS algorithm in the same network as in Figure 4.3

placements described in previous section (Section 3.5.2). The results are shown in from Figure 4.5 to Figure 4.10. In these figures, each error bar represents one standard deviation. Each data point represents to five repetitions.

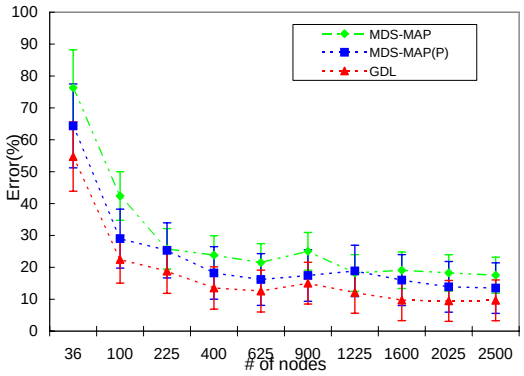


Figure 4.5: Comparison of the accuracy of localization ($r = 2$m)

From Figure 4.5 we can see that when r=2, the accuracy of our algorithm is higher than the accuracy of MDS-MAP series algorithms except when $n < 100$. This is because in a network of $r \leq 2$ and $n < 100$, at least 90% of all nodes in the 36-node network can reach with each other in one hop—this means that the diameter of a network with $r \leq 2$ meters and n=36 is just two hops. This situation is the only one in which we found our algorithm does not work well.

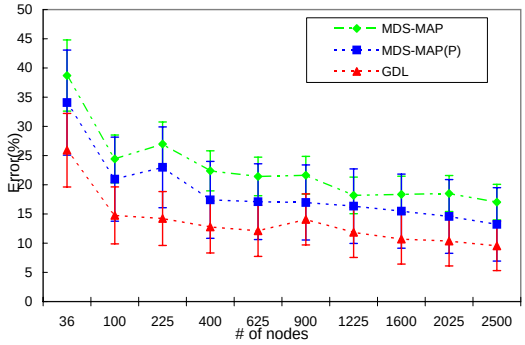


Figure 4.6: Comparison of the accuracy of localization ($r = 4$m)

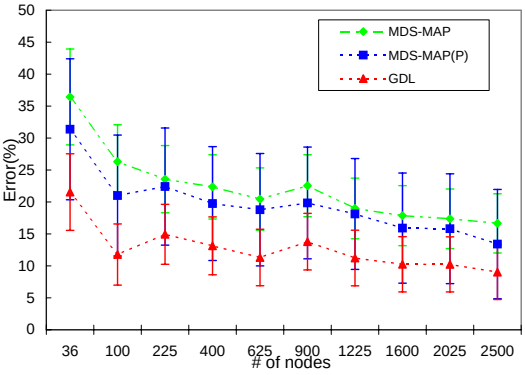


Figure 4.7: Comparison of the accuracy of localization ($r = 6$m)

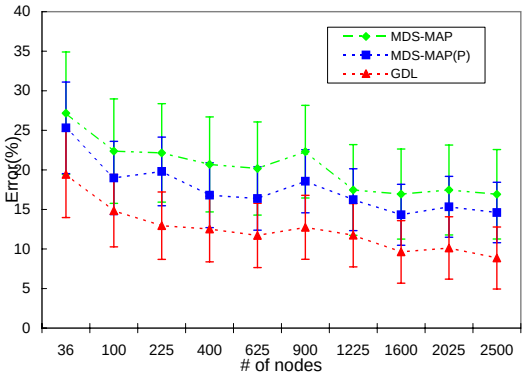


Figure 4.8: Comparison of the accuracy of localization ($r = 8$m)

From Figure 4.5 to 4.8, we can see that when $4 \leq r < 10$ meters, the improvements of accuracy in localization with our algorithm are stable in the networks tested from a small scale (36 nodes) to the largest scale (2500 nodes), comparing with the MDS-MAP series algorithms.

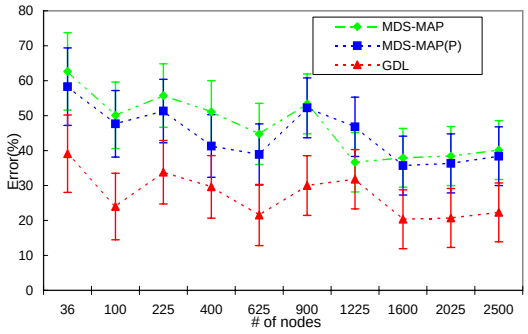


Figure 4.9: Comparison of the accuracy of localization ($r = 10$m)

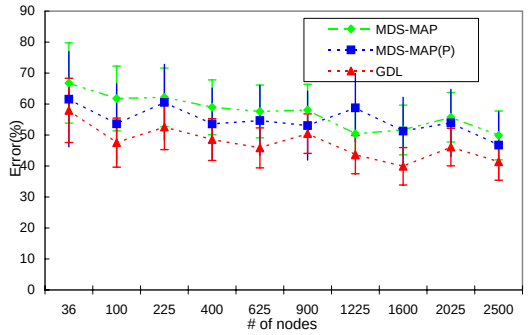


Figure 4.10: Comparison of the accuracy of localization ($r = 12$m)

Our technique does not require each node to connect to more than a certain number of neighbor nodes, but when $r \geq 10$m, many nodes in the network have only one or two connections to other nodes. Figure 4.9 and Figure 4.10 shows that in such a situation (when $r < 10$m) our algorithm will offer no significant improvement in accuracy over the MDS-MAP series algorithms.

Based on these results, we can compute the overall average accuracy corresponding to the various grid sizes (r) tested. These results are shown in Figure 4.11, 4.12, 4.13. We

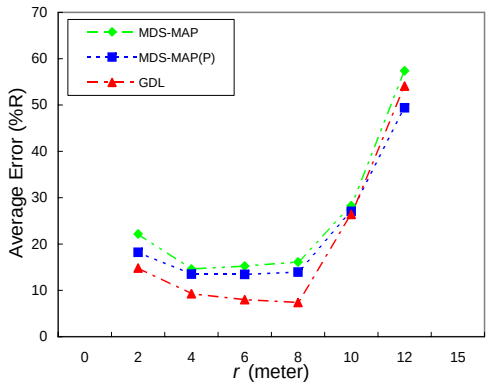


Figure 4.11: Comparison of overall average accuracy of localization ($n = 36 - 2500$)

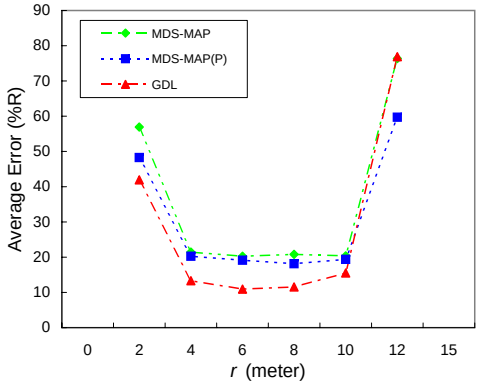


Figure 4.12: Comparison of overall average accuracy of localization ($n = 36 - 100$)

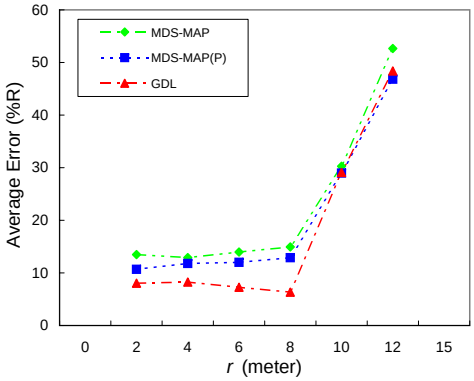


Figure 4.13: Comparison of overall average accuracy of localization ($n = 225 - 2500$)

can see from Figure 4.11 that when $r \geq 10$m, the error in the location generated by our algorithm is similar to that of the comparison algorithms. When $r \leq 2$m, the improvement in the accuracy from our algorithm is not as same as when r has greater values. Since the application of the proposed algorithm was shown above to be poor when $r \leq 2$m and $n \leq 100$, as shown in Figure 4.12, we calculated overall results, shown in Figure 4.13, without this segment. Figure 4.13 shows that our algorithm improves the accuracy of localization about 45% and 36% respectively compared with the MDS-MAP and MDS-MAP(P) algorithms using only hop-counting, when $2m \leq r < 10$m; these figures reduce to about 34% and 26% respectively compared with the MDS-MAP and MDS-MAP(P) algorithms, when $2m \leq r \leq 12$m.

4.4 Summary and Conclusion

This chapter proposes a Geographic Distributed Localization (GDL) algorithm based on hop-counting. In addition to calculating hop numbers from bootstrap nodes, our method stores local network connections for a node by generating special "hop coordinates" that augment the hop number with additional information. Using a simple voting mechanism, our method generates a center node geographically distributed for each local area, each of which then creates a local relative map for its area. By computing and transmitting transformation matrix between adjacent center nodes, our algorithm avoids the merge process required by MDS-MAP(P). Extensive simulation shows that our algorithm improves the accuracy of localization by about 45% and 36% respectively compared with the MDS-MAP and MDS-MAP(P) algorithms. It also has a lower communication cost $O(n)$ for whole network, and has a low constant memory cost per node. Our simulation shows that the algorithm is effective over a wide range of cases, although in two specific cases, namely (1) when n is small and all nodes are close with each other (r is small) and (2) when all

nodes are far away from each other (r is large), the proposed algorithm may not be helpful.

For future research, as more and more commercial sensor networks provide RSSI, it is worth noting that it is possible for our algorithm to be combined with RSSI to improve the accuracy of localization further. Also, while research has shown that the MDS method is suitable for anchor-free localization in WSNs, our algorithm can also be adapted to other emerging methods such as SDP (Semi-Definite Programming) [87].

4.5 Appendix: MultiDimensional Scaling (MDS)

MultiDimensional Scaling (MDS) is a set of data analysis techniques that display the structure of distance-like data as a geometrical picture [11]. MDS finds a placement of points in a low-dimensional space for a set of objects, from the matrices which include distance, similarities or proximities between the objects.

MDS is a generic term that includes many different specific types. These types can be classified according to whether the similarities data are qualitative (called nonmetric MDS) or quantitative (metric MDS) [20].

Because nonmetric MDS focuses on qualitative analysis, while our location estimation asks for the quantitative result of location, we focus on metric MDS.

In the category of metric MDS, the number of similarity matrices and the nature of the MDS model can also classify MDS types. This classification yields classical MDS, in which similarity data are in one matrix and the model of MDS is unweighted, replicated MDS, in which similarity data are in multiple matrices and the model of MDS is unweighted, and weighted MDS, in which similarity data are in one or multiple matrices and the model of MDS is weighted. We discuss the classical-replicated-weighted classifications in the following sections.

4.5.1 Classical MDS

The identifying aspect of Classical MDS (CMDS) is that there is only one similarity matrix. Table 4.1 is a matrix of similarity data suitable for CMDS.

Distance	Node1	Node2	Node3	Node4
Node1	0	9.94	4.32	7.29
Node2	9.94	0	14.06	14.94
Node3	4.32	14.06	0	11.02
Node4	7.29	14.94	11.02	0

Table 4.1: Pair distance matrix for 4 nodes (m)

It contains the distances between 4 nodes marked as “1”, “2”, “3” and “4”. The nodes are the “objects” and the distances are the “similarities”. A MDS of these data gives the picture in Figure 4.14.

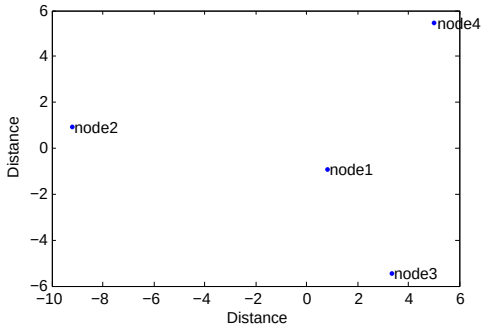


Figure 4.14: A result of MDS given the data from Table 4.1, here X, Y axes are in meters.

This map has 4 points, one for each of the 4 nodes. Nodes that are similar (have short distance) are represented by points that are close together, and nodes that are dissimilar (have large distance) by points far apart.

4.5.2 Replicated MDS

The Replicated MDS (RMDS) is interested, if we consider that a sensor can do multiple measurements through different measurement techniques in order to compute its location.

But since in this dissertation, we are focusing more on localization which only uses single measurement technique, only one matrix of distance can be generated by the single measurement technique. Therefore, it is not necessary to apply replicated MDS in our algorithm.

4.5.3 Weighted MDS

The next major MDS development, Weighted MDS (WMDS), generalizes the distance model so that several similarity matrices in WMDS could be assumed to differ from each other in a systematically nonlinear system. Whereas RMDS only accounts for individual differences in response bias, WMDS incorporates a model to account for individual differences in the fundamental perceptual or cognitive processes that generate the responses.

Since in this dissertation, we do not consider the difference between distance measurement techniques. So, we will not talk about weighted MDS in detail. We will discuss the possibility of using weighted MDS and replicated MDS in localization in Chapter 7.

4.5.4 Implementation of Classical MDS

We only applied classical MDS in our algorithm, so, here we only talk abut how to implement the classical MDS. In classical MDS the proximities are treated directly as distances. However, the matrix of (dis)similarities P should be preprocessed in order to have a metric. Two properties have to hold: *non-degeneracy* and *triangular inequality*. Non-degeneracy means that $d_{i,i} = 0$ for all i and the triangular inequality states that $d_{i,j} + d_{j,k} \geq d_{i,k}$ for all triplets (i, j, k). The matrix obtained after preprocessing is labeled D. Then the matrix of scalar products is:

$$B = -\tfrac{1}{2}\left[I - \tfrac{1}{n}ii'\right] D^2 \left[I - \tfrac{1}{n}ii'\right] \tag{4.7}$$

where I is an n by n identity matrix and i a unity vector of length n. This matrix B is symmetric and positive semidefinite. Performing the Singular Value Decomposition (SVD)

$$B = VAV^{'} \tag{4.8}$$

We can define $B = XX^{'}$ with $X = VA^{1/2}$ being the matrix of coordinates. The eigenvectors are supposed to be normed. Retaining only the first r eigenvectors leads to a solution in lower dimensionality.

It can be shown that the elements of the double centered dissimilarity matrix D equal minus two times the scalar products:

$$d^2_{(i,j)} - \frac{\sum_{j=1}^{n} d^2_{(i,j)}}{n} - \frac{\sum_{i=1}^{n} d^2_{(i,j)}}{n} + \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} d^2_{(i,j)}}{n} = -2\sum_{a=1}^{m} x_{(i,a)}x_{(j,a)} \tag{4.9}$$

This implies that the summation over a in that runs over 1 to r instead of m. This implies that the first r eigenvectors of X are the best lower-rank approximation in the least-squares sense.

Chapter 5

Anchor-Free Localization in (n, m) WSNs

In this chapter, we consider how to localize individual nodes in mixed WSNs when some subset of the network nodes can be in motion at some given time. For situations in which it is not practical or cost-efficient to use GPS or anchor nodes, we propose a Mixed Geographic Distributed Localization (MGDL) algorithm for mixed WSNs. Taking advantage of embedded sensors (accelerometers) that are present in standard motes, MGDL estimates the distance moved by each node. If this distance is over a threshold, then the moved node will trigger a series of mobile localization procedures to recalculate and update its location. Such procedures will be stopped when the node stops moving. Data collected using Tmote Invent nodes (Moteiv Inc.) and simulations show that the proposed detection method can efficiently detect the movement, and that the localization is accurate and the communication is efficient in different static and mobile contexts.

5.1 Introduction

When we talked about the localization in $(n, 0)$ WSNs, the mobility of nodes is not an issue considered in the algorithm in Chapter 4. But as we talked in Chapter 1, mobility is a common feature for many WSNs. Theoretically, we can consider all WSNs as mixed WSNs, no matter they are mobile or static network at all. So, one important problem for localization is how to locate a node's position, and keep its location updated over time.

Though many localization algorithms have been proposed for wireless ad hoc networks or wireless sensor networks [3, 5, 10, 14, 15, 19, 82, 83], they assume that the nodes inside of the networks are static. Little research has been presented on considering localization in cases where the network cannot be assumed to be static.

There are several potential solutions to provide localization in mixed WSNs while considering mobile nodes existing inside of them:

(1) Let mobile nodes deploy expensive global positioning systems (GPS) to get updated location. One problem for this solution is that many applications require sensor network mobility are in the environment where GPS signals may not be available. One example is fire fighter example mentioned before: since the mobile nodes that are carried by fire fighters will often be deployed inside of buildings, where GPS signal is not available, so such a GPS solution is not feasible.

(2) Re-execute current static localization algorithms periodically to keep the location for mobile nodes updated. If some nodes are moving fast, such static localization algorithms will need to be restarted frequently to approach reasonable performance. If restarting the localization algorithms is done too frequently, it will have a significant energy and communication cost for localization over the entire network. So such solution is not feasible too.

(3) Redesign localization algorithms that are particularly focused on the problem of

localization of mobile nodes. There are some existing mobile localization schemes, such as MCL [31], MCB [4] and ELA [92]. They rely on an assumption explained below, that anchor nodes are available and can be used for localization. This assumption is problematic in many settings, particularly in the case where anchor nodes cannot be guaranteed to be fixed at known locations. Our work focuses on finding a solution in this category that does not require any anchor nodes in computation.

Because they assume that there are some anchor nodes that are aware of their physical locations exactly even when moving, MCL, MCB and ELA focus on ways to use these mobile anchor nodes to provide localization information to nearby nodes. There are several problems that may be faced under these localization algorithms: 1. The accuracy of AB (Anchor-Based) schemes is related to the number of anchor nodes, so, in order to achieve high accuracy, AB schemes usually need large sets of anchor nodes. 2. The fact that AB schemes depend heavily on anchor nodes makes them vulnerable to the loss or malfunctioning of any of these anchor nodes. 3. Current AB solutions such as MCL, MCB and ELA need all nodes, or at least all anchor nodes, to broadcast their locations periodically, even when there is no node movement.

This chapter talks about an AF localization called MGDL for mobile WSNs. We assume that a network, which MGDL is running on, is comprised mostly of low-mobility nodes that are embedded into the environment, while some other nodes are carried by some mobile objects.

Based on the above assumption, MGDL generates a distance measurement for each node by combining local network connection information with hop numbers generated without the help of GPS or anchor nodes. Using the accelerometers deployed in standard motes, MGDL monitors a moving distance for each node after the whole network is started, then with a probe procedure to detect any mobility of the node. If movement is detected, then the moved node will trigger a series of procedures to recalculate/update the estimated

location for that moved node locally. This procedure will continue while movement continues to be detected.

In this chapter, we make the following contributions: (i) We design a movement detection procedure for standard motes in WSNs to detect the movement of a node by using accelerometers. (ii) We propose a MGDL localization algorithm in detail, based on (i). Simulation on the algorithm shows that the MGDL has high accuracy in localization in different placements of networks, as well as an efficient communication cost while facing different situations of mobility.

The remainder of the chapter is organized as follows. Section 5.2 discusses the detail of our Mixed Geographic Distributed localization algorithm; Section 5.3 reports simulation results. Finally, Section 5.4 gives our conclusions.

5.2 Mixed Geographic Distributed Localization (MGDL)

In this section, we introduce the Mixed Geographic Distributed Localization (MGDL) algorithm in detail.

5.2.1 Overview of MGDL Algorithm

We assume that a node in a WSN can be either in a mobile or a static state, and in order to distinguish the state of a node, we use "mobile" or "static" to distinguish whether a node is mobile or not. At the same time, we use "updated" or "non-updated" to distinguish whether a node is localized or not. Combining these labels gives four states, which are represented as `S/N` ("Static/Non-updated"), `S/U` ("Static/Updated"), `M/N` ("Mobile/Non-Updated") and `M/U` ("Mobile/Updated"), which are shown in Table 5.1.

Based on the assumption that each node in the networks stays in the "S/N" state initially, MGDL uses a technique similar to hop-counting as a measurement procedure (Section 3.4)

States of a Node	Description
S/N	the state of a node is Static and its location is Not updated
S/U	the state of a node is Static and its location is Updated
M/N	the state of a node is Mobile and its location is Not updated
M/U	the state of a node is Mobile and its location is Updated

Table 5.1: The states of a node

to measure the distance from some bootstrap node to other nodes. After the running of the measurement procedure, each node will collect a set of hop coordinates from its neighbor nodes that are within one (or k) hop(s) distance of itself, and then it will run Dijkstra's algorithm to get the shortest path between each pair of nodes. After that, it will construct a local map (Section 5.2.3). A transformation procedure (see Section 5.2.4) will merge the local map inside of each node into a global map. After all the above procedures are done, the state of the node will be set as "S/U".

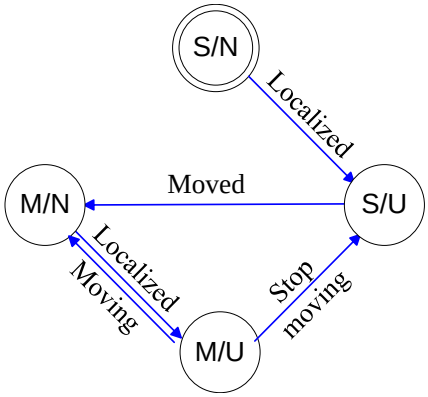


Figure 5.1: The state diagram of a node

In order to detect the movement of a node, we utilize an accelerometer, which is a standard component in many current motes (such as Moteiv's Invent [39]), to detect the movement of the node. The detail of movement detection using accelerometer was introduced in Section 5.2.5. If there is a node that is starting to move inside of the network, the accelerometer can detect its acceleration, and then our algorithm will compute the distance moved based on the acceleration. If this distance is beyond a threshold, then the state of

the node will be set as “M/N”, and mobile localization procedures (Sections 5.2.3, 5.2.6 and 5.2.7) will be triggered to resample hop-coordinates again, to recompute the local map and transformation matrix for that local map to merge this map into the global map. While movement continues to be detected, the above procedures will continue. Once the movement is no longer detected, the above updating procedures will be stopped and the state of the node will be set back to “S/U”. Figure 5.1 shows the states for a node.

Figure 5.2 shows the states and the whole MGDL algorithm for a node.

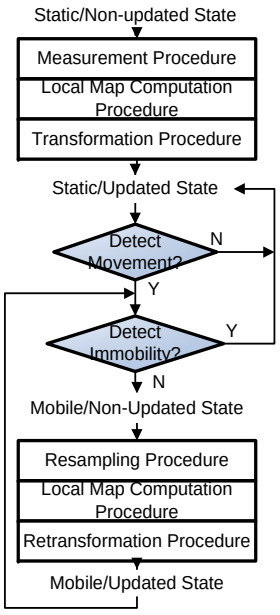


Figure 5.2: Sensor node states and MGDL algorithm

5.2.2 Measurement Procedure

In the measurement step, we are reusing the hop-coordinates technique for our algorithm. Since it is already introduced in Chapter 3, here we do not give the detail.

Suppose that an arbitrary node a is calculating its hop distance, and node b is one of the neighbors of node a. If this node a is in “S/N” state, then the basic hop-coordinates procedure, which is shown in Section 3.4, runs in node a (All operations are shown in

Procedure 10). A bootstrap node is necessary and should be pre-assigned before the whole MGDL procedure runs. In practical, we can either use a sink node, which is a gateway used to connect WSNs with outside system, to bootstrap this step, or use a method proposed in Section 4.2.1, whose idea is to mix b constant candidate nodes for bootstrap in a n node WSN, then select bootstrap nodes X and Y from these b bootstrap candidate nodes. Deterministic selection of two bootstrap nodes, specified as X and Y, are most far away a pair of nodes among these b candidates for bootstrap. In simulation, we use the method proposed in Section 4.2.1, and use node X as a bootstrap node for this step.

Procedure 10 Measurement Procedure in node a

```
1: if the state of node a ==“S/N” then
2:     Call Procedure 2
3: end if
```

In this step, we also assume that each node is in static status. Since this procedure is a one-time procedure and must run before other procedures, so we can do it right after nodes deployment and keep each node be in static status. Mobile nodes do not have higher cost than static nodes does in this step. So, the total cost for this step is as follows (suppose in node a): a computational cost of $O(1)$ per node, a communication cost of $O(N_a)$ per node, a memory cost of $O(N_a)$ for each node; and a communication cost of $O(n)$ for the whole network.

5.2.3 Local Map Computation

In this step, each node will compute a local map for its neighboring nodes based on the hop-coordinate computed in the previous step. After the generation of hop-coordinates discussed in Section 5.2.2, each node will send a request to its neighboring nodes that are within k hops to send back their hop coordinate from some bootstrap node.

After each node receives the hop coordinate from its neighbors, that node will compute

shortest paths between all pairs of nodes in k hops to that node, using Dijkstra's algorithm or other similar algorithms, the output is a ($|N_a|$+1)×($|N_a|$+1) shortest path matrix (here $|N_a|$ is the number of nodes that can be reached by node a in k hops).

Then, we apply classic MDS (talked in Section 4.5) to this ($|N_a|$+1)×($|N_a|$+1) shortest path matrix and retain the first two largest eigenvalues and eigenvectors to construct a 2-D local map. The coordinates of the nodes in this local map are stored in an estimated coordinate matrix $F_a = [..., f_j, ...]$ in node a, where f_j is an estimated coordinate for node j in the neighboring area of node a ($j \in N_a \cup a$), the coordinate system of this local map (represented as $F_(a)$) are normalized only for this local area by MDS.

The total cost for this step is a computational cost of $O(|N_a|^3 n)$ and a memory cost of $O(|N_a|^2)$ per node, with $O(1)$ per node communication cost to retrieve the hop coordinates from its neighboring nodes in this step.

5.2.4 Transformation Procedure

This step will run if a node a is in static and non-updated ("S/N") state. In this step, we will continue the localization process to assemble the local maps that are computed and stored in each node into a global map through a transformation.

First, we will start from some node (in our simulation, we use node Y selected in Section 5.2.2) to compute one transformation matrix for each of its neighboring nodes. Then node Y will send the corresponding transformation messages to its neighbors to let them start to compute the transformation matrix with their neighbors, and continues this procedure until all nodes in the network are transformed.

Then, after we find the set of neighboring nodes, we compute a transformation matrix for each node b in the neighboring node set N_a of node a.

Suppose that there are two sets of neighboring nodes, N_a and N_b, that are within k hops

of the nodes a and b, respectively. Their intersection is $I = N_a \cap N_b$, and we use matrix $I_a = \left\{\ldots, (x_i, y_i)', \ldots\right\}'$, where $i \in I$, (x_i, y_i) are the coordinates of one node $i \in N_a$ in the local map F_a generated by node a. And similarly $I_b = \left\{\ldots, (\grave{x}_i, \grave{y}_i)', \ldots\right\}'$, Where $(\grave{x}_i, \grave{y}_i)$ are the coordinates of a node $i \in N_b$ in the local map F_b generated by node b. We can then compute a transformation matrix T such that it minimizes

$$\sum \sqrt{((x_i - \breve{x}_i)^2 + (y_i - \breve{y}_i)^2)} \tag{5.1}$$

where $(x_i, y_i) \in I_a$, $(\grave{x}_i, \grave{y}_i) \in I_b$, and $(\breve{x}_i, \breve{y}_i) \in I_b \times T$.

The procedure is given in Procedure 11.

Procedure 11 Compute transformation matrix *T* for N_a in node a

1: **if** the state of node a !="S/N" **then**
2: RETURN
3: **end if**
4: set *self* as "transformed"
5: **for** each node $b \in N_a$ **do**
6: request N_b from node *b*
7: $I = N_b \bigcap N_a$
8: $I_a = ..., (x_i, y_i), , , .,$ such that $i \in I$ and $(x_i, y_i) \in F_a$
9: retrieve I_b from node b
10: compute transformation matrix T_b such that $\sum \sqrt{((x_i - \breve{x}_i)^2 + (y_i - \breve{y}_i)^2)}$ is minimized, here $(x_i, y_i) \in I_a$, $(\breve{x}_i, \breve{y}_i) \in I_b \times T_b$.
11: send matrix T_b to node *b*
12: **end for**
13: set the state of *self* as "S/U"

If a node receives a transformation matrix *T*, then it will first check whether it is already transformed or not; if so, the node will drop such a message, and if not, it will apply Procedure 11. This will allow the node to compute the local map (which will be a portion of the global map), which it will then send back to its $|N_a|$ neighboring nodes. At the same time, the node will find its neighboring node set, and will compute transformation matrix T for each node in the set using Procedure 12.

Procedure 12 Receive transformation matrix T in node a

1: INPUT: matrix T_a from some neighboring node
2: **if** the state of node a !="S/N" **then**
3: RETURN
4: **end if**
5: **if** this node is transformed **then**
6: drop (message T_a)
7: RETURN
8: **end if**
9: transform the local map F_a with T into a portion of global map H
10: send back corresponding coordinates in the local map to the neighboring nodes in N_a
11: set *self* as "transformed"
12: **for** each node $b \in N_a$ **do**
13: request N_b from node *b*
14: $I = N_b \bigcap N_a$
15: $I_a = ..., (x_i, y_i), , , .,$ such that $i \in I$ and $(x_i, y_i) \in F_a$
16: retrieve I_b from node b
17: compute transformation matrix T_a such that $\sum \sqrt{((x_i - \check{x}_i)^2 + (y_i - \check{y}_i)^2)}$ is minimized, where $(x_i, y_i) \in I_a$, $(\check{x}_i, \check{y}_i) \in I_b \times T_a$.
18: send matrix T_b to node *b*
19: **end for**
20: set the state of node a to "S/U"

After we have flooded the network to the transformation step in the whole networks, we achieve the global map for the whole network, which is stored in a distributed way in each node in the network.

Total cost for this step: computational cost of $O(|N_a|)$, memory cost of $O(|N_a|)$ for nodes, and communication cost of $O(1)$ for each node or $O(n)$ for whole network.

5.2.5 Mobile Measurement Techniques

Until now, we have only discussed how to do measurement/localization without considering the mobility of nodes. (After the previous procedures, current state for each node is S/U). When mobility is taken into consideration in localization systems, the first question that comes up is how the state of a node can be known. This means to the localization must know whether this node has moved or not, whether the node has stopped moving, or how fast the node travels when it is moving. This section describes how we take advantage of embedded sensors, called accelerometers that are present in standard motes. Using these sensors we estimate the distance moved by each node. Based on this distance we try to figure out the status of a node: whether it has moved or not, whether it has stopped moving, or how fast it can move.

In order to detect the movement of mobile nodes, we need some sensor that can detect and quantify node movement. There are several sensors that could be potential candidates for use in WSNs. Pyroelectric InfraRed (PIR) [91] sensor detect infrared rays that are emitted by live objects such as animals or human beings, in order detect the object's motion. But because a sensor node cannot emit strong infrared beams, the PIR sensor is not a good choice for us. Compass [21] can only be used to detect the direction of an object's movement, while we want to detect three dimensional motion for each sensor node.

2D and 3D Accelerometer

In our algorithm, we make use of accelerometers installed in standard nodes to detect node movement. An accelerometer is a device that measures its own acceleration. We can use a 2D (X-Y) accelerometer to measure 2D acceleration, or a 3D (X-Y-Z) version to measure 3D acceleration. The component we used for 2D in this section is a standard component in current commercial motes, such as the Moteiv Inventor mote [39]. In order to detect 3D movement of the node, one can install an inexpensive external 3D accelerometer, such as the MMA7260Q accelerometer [38] from Freescale.

Movement Detection

With the aid of a 2D accelerometer, we can roughly measure the acceleration vector $\vec{a}$ in a plane. Since the accelerometer cannot detect the rotation of a node, it is of limited use as a direct way to measure position changes. Therefore, we use the integral of the absolute value of $\vec{a}$ to compute an approximation of the moved distance $d = \int\int |\vec{a}|d^2t$. If such distance is beyond a threshold, we conclude that this node has moved.

First we assume that at the beginning of time, $t = 0$, every node inside of the network is still. Consider an arbitrary node with acceleration $\vec{a} = 0$, velocity $v = 0$ and the distance it has moved as $d = 0$. We, then, periodically sample the accelerometer in that node. We assume that the interval time for sampling is dt, and the reading of acceleration from the accelerometer in that node is $\vec{a}$. So, current velocity for this node can be approximated as $v = \int |\vec{a}|dt$, and the distance moved from the beginning location (when $t = 0$), can be approximated as $d = \int\int |\vec{a}|d^2t$. If d is beyond a threshold ϵ, we then say this node has be moved We then set $v = 0$ and $d = 0$ and restart the measurement. Thus, though the values of $\vec{a}$, v and d are not accurate, in comparison to their real values, they are sufficient to detect the movement of a node. The complete process is described in Procedure 13.

Procedure 13 Movement Detection Procedure

Require: this procedure will be invoked to read the accelerometer during the time period dt.

1: $v =$ previous $v + \int |\vec{a}|dt$
2: $d =$ previous $d + \int vdt$
3: **if** $d >$ threshold ϵ **then**
4: $v = 0$
5: $d = 0$
6: set the state of *self* as "M/N"
7: Return "detect movement"
8: set *self* as "untransformed"
9: **end if**

If Procedure 13 detects that a node has moved, then the localization algorithm in that node will recompute its location. The threshold ϵ will decide when a mobile node should recompute its location. The smaller the threshold is, the higher the frequency with which the node computes its location, leading to a higher localization accuracy as well as higher communication cost.

In order to evaluate the procedure to detect the movement of a node, we set up an experimental environment in a long hallway of about 50 m. A node carried by a person walks as a mobile node. The above movement detection procedure is running inside of that node with $dt = 0.1$s. At first, we will let the mobile node start to move from one end of the hallway. We allow it to move for 5 seconds at walking speed, then stop for 5 seconds, and then move again and so on until the end of the hall is reached. The sensor is then returned down the hallway along the same path in the opposite direction. The result of the first 40 seconds of this experiment is shown in Figure 5.3.

From Figure 5.3, we can see that this movement detection procedure can detect the movement of a node when the node is moving, albeit with an average delay of about 0.95 s.

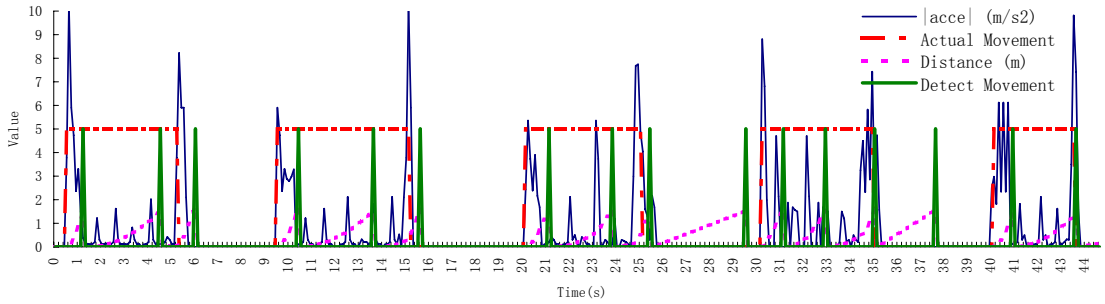


Figure 5.3: First 40 seconds of experiment on movement detection with a 2D accelerometer. Here, movement is detected within each movement block, albeit with a slight lag, and is only occasionally spuriously detected between movements. The $dt = 0.1$s and threshold ϵ = 1.5m. If a node is moving then actual movement will be shown as value = 5m, otherwise actual movement will be shown as value = 0.

Immobility Detection

After we detected the motion of a node, we assume it remains in motion until we can detect that the node has become immobile.

It might seem that we could simply read the accelerometer to detect the immobility of a node. But, the problem is that an accelerometer cannot detect the rotation of a node. If a node rotates, the acceleration vector $\vec{a}$ measured by the accelerometer may not reflect the real acceleration vector for that node. So, integrating the measurement from an accelerometer alone may not be enough to measure the real movement vector. An experimental result is shown in Figure 5.4.

Hence, we do not use an accelerometer to detect whether a node is immobile or not. Instead, we use the idea of comparing the previous hop-coordinate of the same node with the resampled hop coordinates of itself. Assuming there is an arbitrary node a, if the previous hop-coordinate hop_a is the same as the current hop-coordinate hop_a, we say that the node's neighboring nodes N_a do not move, and therefore node a is immobile.

Even in a group movement, mobile nodes read hop-coordinates from static nodes for resampling. If such a mobile node is still moving, it will create different hop-coordinates

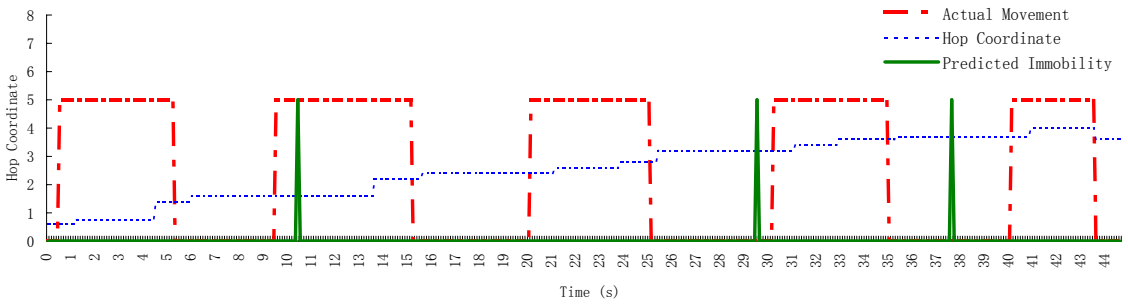


Figure 5.4: First 40 seconds of experiment on immobility detection with hop-coordinates

for itself. Therefore, if the hop coordinates created by the resampling procedure is not changed and no movement is detected by the accelerometer, we can conclude that the node is immobile, and change the state of this node from "M/U" to "S/U", if right now the sate of node is "M/N", then at first this node will run the procedures in Section 5.2.6 and 5.2.7 to update this node's location, then change its state to "S/N".

One main problem, which is different to the localization in static WSNs, is the ability to detect the mobility of nodes inside a WSN. With the help of accelerometer, which is a standard component in current commercial sensor nodes, we developed methods to detect both movement and immobility of a node. Though the detection rates of our methods for both movement and immobility are not one hundred percent accurate, we believe that our work is a valuable first step in trying to include sensor components into localization algorithms.

5.2.6 Resampling Procedure

If we detect that a node is moving with the movement detection procedure (Procedure 13 in Section 5.2.5), the movement detection procedure will mark the state of this node as "M/N" (the node is Mobile but its location is Non-updated). Then, we will update its location with the procedures introduced in this section and Section 5.2.3 (Local Map

Procedure 14 Resampling Procedure for Node a

1: **if** the state of node a !="M/N" **then**
2: RETURN
3: **end if**
4: **if** $|N.static_a| == 0$ **then**
5: Return the previous hop-coordinates $hop.coor_a$ in node a
6: **else**
7: **for** each node $b \in N.static_a$, for which the state of node b is "S/U" **do**
8: request $hop.coor_b$ from node b
9: **end for**
10: $hop.coor_a = \frac{\sum_{b \in N.static_a} hop.coor_b}{|N.static_a|}$
11: **end if**
12: Return $hop.coor_a$

computation) and 5.2.7 (Retransformation). The resampling procedure samples the hop-coordinate for this node from its neighboring nodes again. While, we only study the mixed WSNs, in which there are much more static nodes than mobile nodes. In order to increase the accuracy of resampling, we only get hop-coordinates from nodes that are marked as static (in "S/U" state), instead of from nodes marked as mobile (in "M/N" or "M/U" states). The whole procedure is described in Procedure 14.

Here, $O(N.static_a)$ is a set of node a's neighboring nodes with "S/U" state. The total cost for this step is as follows: computational cost of $O(1)$, communication cost of $O(N.static_a)$, memory cost of $O(N.static_a)$ for each node.

5.2.7 ReTransformation Procedure

After the resampling procedure and local map computation (which is as same as in Section 5.2.3), we transform the local map inside this moved node into a portion of the global map. In this process we get a new transformation matrix for this local map, since the old transformation matrix is out of date. The retransformation procedure is different to the transformation procedures 12 or 11: The transformation procedures in Section 5.2.4 are generating transformation matrices for its neighboring nodes, while retransformation

procedure is generating a transformation matrix for itself.

Suppose this moved node is node a. Since our idea is using overlapped neighboring nodes between two nodes (one has a transformation matrix, another one has not) to generate a transformation matrix, it is more accuracy in general if there are more overlapped neighboring nodes shared between these two nodes. Here, we find a closest neighboring node with "S/U" state (here we assume that it is node b); then compute a new transformation matrix T for node a.

Suppose that there are two sets of neighboring nodes N_a and N_b for nodes a, b, respectively. Their intersection is $I = N_a \cap N_b$, and we use matrix

$$I_a = \left\{\ldots, (x_i, y_i)', \ldots\right\}' \tag{5.2}$$

where $i \in I$, (x_i, y_i) are the coordinates of one node $i \in N_a$ in the local map F_a generated by node a. And similarly

$$I_b = \left\{\ldots, (\grave{x}_i, \grave{y}_i)', \ldots\right\}' \tag{5.3}$$

Where $(\grave{x}_i, \grave{y}_i)$ are the coordinates of a node $i \in N_b$ in the portion of global map H already generated by node b, which equals $F_b \times T_b$. We can then compute a transformation matrix T for node a such that it minimizes

$$\sqrt{\sum((x_i - \check{x}_i)^2 + (y_i - \check{y}_i)^2)} \tag{5.4}$$

where $(x_i, y_i) \in I_a \times T$, $(\grave{x}_i, \grave{y}_i) \in I_b$, and $(\check{x}_i, \check{y}_i) \in I_b$. The procedure is given in Procedure 15.

Here, $O(N.static_a)$ is a set of node a's neighboring nodes with "S/U" state. Total cost for this step: computational cost of $O(|N.static_a|)$ per node, memory cost of $O(|N.static_a|)$ per node, and communication cost of $O(1)$ for each node or $O(n)$ for the whole network.

Procedure 15 Recalculate Transformation Matrix T for Node a

1: find a node b from $N.static_a$, such that $|hop.coor_a - hop.coor_b|$ is minimized.
2: request $N.static_b$ from node *b*
3: $I = N.static_b \bigcap N.static_a$
4: request I_b from node *b*
5: compute transformation matrix *T* such that $\sqrt{\sum((x_i - \check{x}_i)^2 + (y_i - \check{y}_i)^2)}$ is minimized, where $(x_i, y_i) \in I_a \times T$, $(\check{x}_i, \check{y}_i) \in I_b$.
6: transform the local map F_a with T into a portion of global map H
7: set this node to M/U state.

5.2.8 Concurrency Discussion in MGDL

In computer science, concurrency [77] is a property of systems in which several computational processes are executing at the same time, and potentially interacting with each other. Our MGDL intends to run in each node simultaneously. It is important to confirm that each node will not be interrupted by other nodes unintentionally.

Besides the movement detection procedure and immobility detection procedure in Section 5.2.5, which only change the states of the node, there are six procedures used in MGDL. To simplification, we use "M" to represent measurement procedures in Section 5.2.2, "C" to represent local map Computation procedure in Section 5.2.3, "T" to represent transformation procedures in Section 5.2.4, "RM" to represent resampling procedure in Section 5.2.6, "RT" to represent retransformation procedures in Section 5.2.6.

After initialized as S/N state, each node needs to run "M", "C" and "T" procedures sequentially. The execution of the procedures in a node (named a) is shown in the first row in Figure 5.5. To synchronize these procedures running in different nodes, in practice, we gave time t_{TIMEOUT} for "M" procedures (described in Section 3.4.2), and time t_{WAIT} after "C" procedure. In the case that one node is running in "C" procedure, while other nodes are running in other procedures, for instance, node a is running in "C" procedure, node b is running in "M" procedure (shown in the second row in Figure 5.5), node a will ignore bootstrap messages (talked in Section 3.4.2) from node b, but still reply neighboring

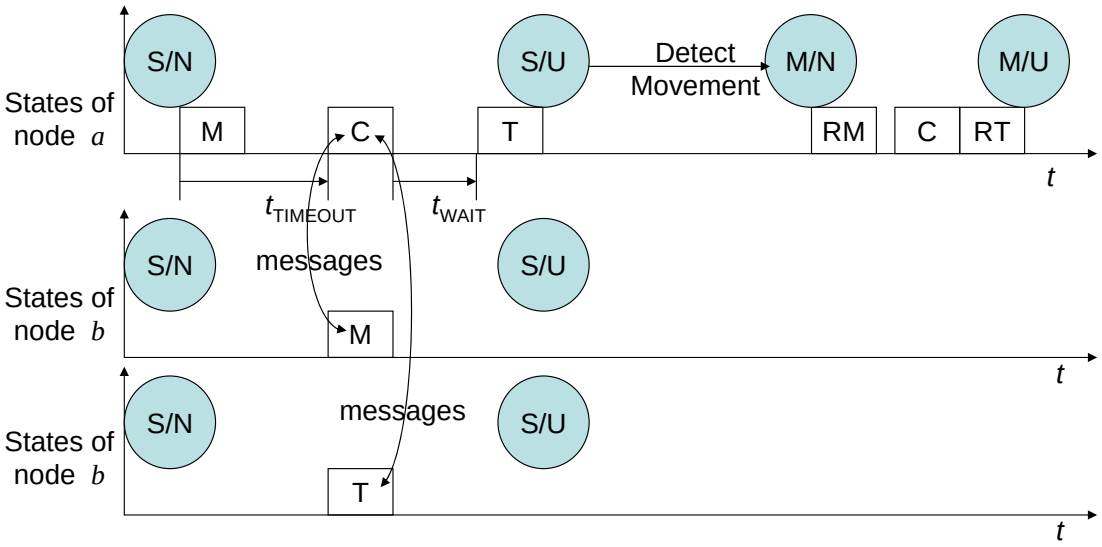


Figure 5.5: The execution of the procedures in node a. Here, the curves with arrows between 1st row and 2nd row and between 1st row and 3rd row mean that there are some communications between node a and node b.

messages (also talked in Section 3.4.2) from node b. The example in the third row in Figure 5.5 shows that there is a node b running "T" procedures, if at this time node a receives messages from node b, since node a did not finish "C" procedure yet, node a will drop the messages from node b.

Since "RM", "C" and "RT" only run when nodes are in M/N state. At that time, usually other nodes already finished "M", "C" and "T" procedures. And the nodes running 'RM", "C" or "RT" procedures only send messages to the nodes in S/U state. So there are no conflicts among nodes running in 'RM", "C" or "RT" procedures.

5.3 Simulation Result

In this section, we will talk about the simulation results for our algorithm.

5.3.1 Simulation Configuration

As same as the implementation of GDL in Chapter 4, we implemented our localization algorithm as a routing agent and our bootstrap node program as a protocol agent in NS-2 version 2.29 [57] with 802.15.4 MAC layer [106] and CMU wireless [28] extensions. The configuration used for NS-2 is RF range = 15 meters, propagation = TwoRayGround, antenna = OmniAntenna.

As same as before, in our experiments, we used uniform placement—n nodes are placed on a grid with $\pm 0.5r$ randomized placement error. Here r is the width of a small square in the grid. We constructed a total of 60 placements with $n = 36, 100, 250, 400, 625, 900, 1600,$ and 2500, and with $r = 2$, 4, 6, 8, 10 and 12 meters, respectively. The reason we use uniform placement with $\pm 0.5r$ error is that usually, such a placement produces both node holes and islands in one placement, as demonstrated in Figure 3.4 (which is shown in Chapter 4). To better simulate realistic mobile network situations in each placement, we let most of the nodes inside of the network behave as static nodes deployed in the environment, while about 10% uniformly randomly selected nodes of the total move inside the network under the following mobility model.

Mobility Model

In order to thoroughly simulate the performance of our algorithm in a WSN, it is necessary to use a mobility model that accurately represents the activities of a mobile node. There are several popular mobility models [17, 6] being in the research community for different scenarios.

The first mobility model is called the "Random waypoint" model [42]. It is one of the most commonly used mobility models for mobile ad hoc networks. Random waypoint model is used to simulate a particular subject's activity. In the random waypoint model,

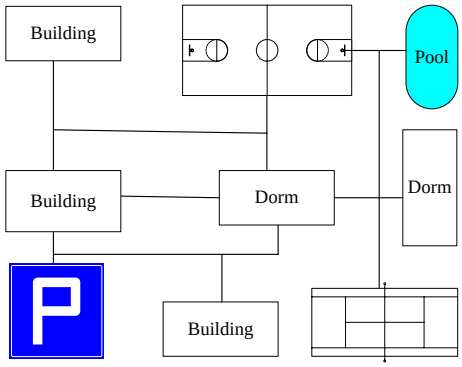


Figure 5.6: An example of pathway mobility model

a node randomly chooses its destination, its speed of movement, and its pause time after arriving at the destination.

The second mobility model is called "Pathway" [6], which is intended to simulate a campus environment. A pathway model predefines a map in the simulation field. The vertices of the graph represent the buildings of the city, and the edges model the streets and freeways between those buildings. Initially, the nodes are placed randomly on the edges of the graph. Then for each node a destination is randomly chosen and the node moves toward this destination through the shortest path along the edges. Upon arrival, the node pauses for a time and again chooses a new destination for the next movement. Unlike the random waypoint model where the nodes can move freely, the mobile nodes in this model are only allowed to travel on the pathways. One example of pathway mobility model is described in Figure 5.6.

Another popular mobility model is the "Manhattan" mobility model, which models movement in an urban area [7]. An example of the Manhattan model is shown in Figure 5.7. In the Manhattan model, the mobile node is allowed to move along the horizontal or vertical streets on the urban map. At an intersection of a horizontal and a vertical street, the mobile node can turn left, right or go straight. The probability of moving on the same street is 0.5, while the probability of turning left is 0.25 and the probability of turning

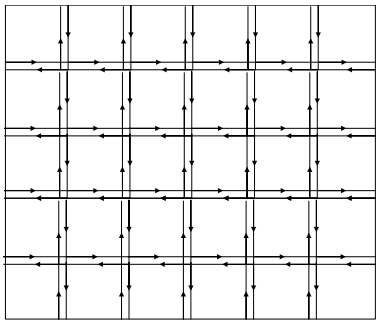


Figure 5.7: An example of Manhattan mobility model

right is 0.25. The Manhattan mobility model focuses on nodes moving along horizontal or vertical streets.

Our primary potential applications for mixed WSNs are for healthcare in assistive environments, which may include different apartment area. Node movement in such environments is complex, and does not follow any tracks. The freeway mobility model intends to simulate traffic that is limited to roads, and the Manhattan mobility model is used to simulate the movements on grids. So, we select the random waypoint model, which allows a node to go anywhere in a network, as a mobility model to simulate our algorithm. In our simulation, the overall pause time equals fifty percent of the total experiment time.

There is no support from NS-2 to simulate an accelerometer in NS-2, so for simplification, we feed the moved distance of one node as an accelerometer's reading to the motion detection procedure in that node.

Each node controlled by the random waypoint model begins the simulation by remaining stationary. It then selects a random destination in the network space and moves to that destination at a speed randomized from a uniform distribution between 0 and some maximum speed. After 25 seconds of movement, a node will be paused for 25 seconds to continue move. In NS-2, there is a file to control the movement of each node, we included all movement information in that file. Upon reaching the destination, the node selects another destination again, and proceeds there as previously described, repeating this behavior

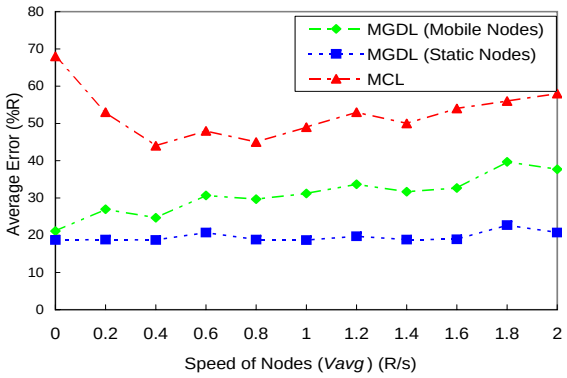


Figure 5.8: Impact of node speed V_{avg} on MGDL and MCL. Here, # of anchor nodes for MCL is about 10% of number of total nodes.

for the duration of the simulation. Each simulation ran for 120 seconds of simulated time.

Simulation Parameters

We will control the following parameters in our simulations:

1. Average speed of nodes (V_{avg}): We represent the speed as the moving distance per time unit. A node's speed is chosen from a uniform distribution [0, $2 * V_{avg}$], so that the average velocity equals V_{avg}.

2. Node Density (ND): The average number of nodes in one hop transmission range. In our placements, we chose $r = 2$, 4, 6, 8, 10 and 12 meters, which corresponds to ND = 144, 38, 17.5, 10, 6.5, and 4.6, respectively.

3. Number of nodes: The total number of nodes inside a WSN.

4. Threshold (ϵ): A threshold is used to judge whether the current node is moving or not. We assume a fixed threshold $\epsilon = 0.1R$ for all simulations (except in varying of ϵ in Section 5.3.3).

5.3.2 Node Speed

Figure 5.8 compares the localization accuracy of MGDL vs. MCL under the varying of V_{avg} from 0 to 1R. Even when the number of anchor nodes for MCL is about 10% of the number of total nodes, we can see that MGDL shows low localization error when nodes are in low-mobility, while MCL is encountering higher error. MGDL achieves an average of 22%R more location accuracy for the overall varying of V_{avg} from 0 to 1R.

5.3.3 Communication Overhead

One important consideration for WSNs is lowering communications overhead. Here, we measure communication overhead with the average number of messages transmitted by a node each second. From Figure 5.9, we can see that the lower the speed of mobile nodes is inside of the network, the lower the communication overhead is in MGDL, while MCL maintains the same communication overhead. This phenomenon shows that our algorithm can efficiently adjust the communication overhead to save more energy when localizing low-speed mobile nodes while keep reasonable accuracy for high-speed mobile nodes, as shown in Figure 5.8. Also, at a given communication speed, we can decrease communication overhead by increasing the threshold ϵ, at the cost of a slower update frequency for mobile localization.

5.3.4 Localization Accuracy

The key metric for evaluating a mobile localization technique is the accuracy of location when nodes are moving. Since MGDL is an AF localization, it does not use anchor nodes, while such anchor nodes will be needed inside of the MCL and ELA. In order to compare MGDL with MCL, and other localization algorithms, we assume that there are only 4 anchor nodes in both MCL and MGDL, in all placements, except as noted. Because at the

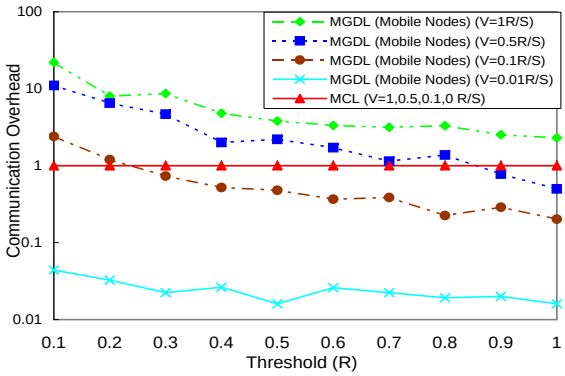


Figure 5.9: Communication overhead with different threshold value on MGDL and MCL under different V_{avg}. Here communication Overhead is measured by the average # of messages per node per second.

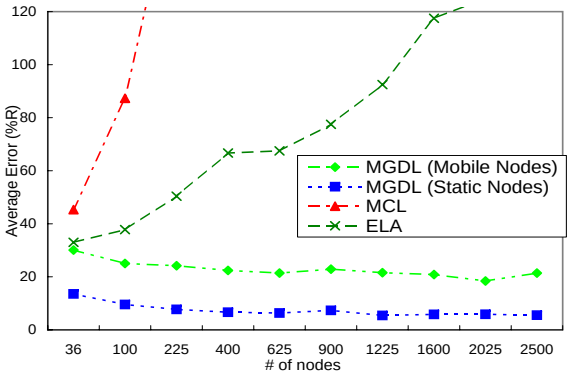


Figure 5.10: Accuracy Comparison with MGDL, MCL and ELA. Here, ND=10, and V_{avg} = 1R/s, Number of Anchors = 4, Number of Nodes = 36,100,225,400, 625,900,1225,1600,2025,2500, threshold ϵ = 0.1R.

same time there are some nodes moving and some nodes being still, we also compute the accuracy of localization for mobile nodes and still nodes, separately.

First we compare these three algorithms under different number of nodes. ELA only has data when ND = 10 available [92]. Figure 5.10 shows the comparison of localization accuracy of MGDL vs. MCL, ELA under different number of nodes = 36 to 2500, when ND = 10. Figure 5.11 compares the localization accuracy of MGDL vs. MCL under different number of nodes with different ND = 144, 38, 17.5, 10, 6.5, 4.6. Both figures share same additional parameters as V_{avg} = 1R/s, and threshold $\epsilon = 0.1R$ for MGDL.

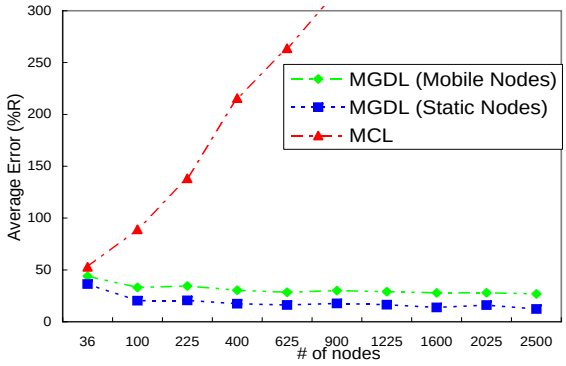


Figure 5.11: Overall accuracy comparison with MGDL and MCL. Here, ND =4.6, 6.5,10,17.5,38,144, V_{avg} = 1R/s, Number of Anchors = 4, Number of Nodes = 36,100, 225,400,625,900,1225,1600,2025,2500, threshold ϵ = 0.1R.

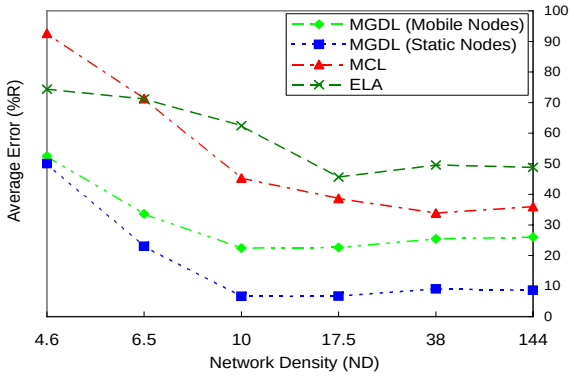


Figure 5.12: Comparison of accuracy vs. density of the networks with MCL. Here, number of nodes is 400 (40 anchor nodes inside them for ELA and MCL), V_{avg} = $1R/s$, threshold ϵ = $0.1R$.

From the figures we can see that MGDL, unlike MCL or ELA, has stable performance on localization accuracy even when the number of nodes is very large, while MCL and ELA usually only have good performance when the number of nodes is small. To achieve good performance with more nodes, it is necessary that they should increase the number of anchor nodes significantly.

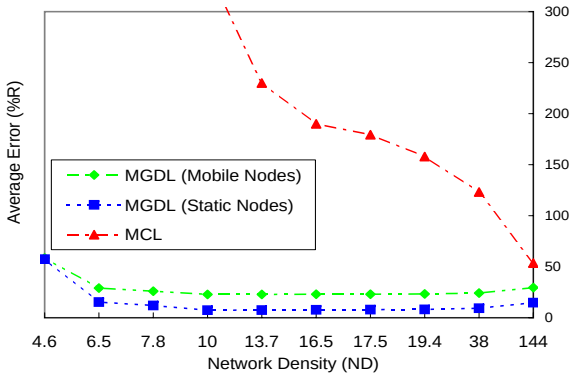


Figure 5.13: Comparison of accuracy vs. density of the networks with MCL. Here, ND =4.6,6.5,10,17.5,38,144, V_{avg} = $1R/s$, Number of Anchors = 4, number of Nodes = 36,100,225,400,625,900,1225,1600,2025,2500, Threshold ϵ = $0.1R$.

5.3.5 Node Density

Since ELA only has data with number of nodes = 400 available for node density, Figure 5.12 shows the comparison of localization accuracy of MGDL, MCL and ELA under number of nodes = 400. Even when there are about 10% anchor nodes inside of the network for ELA and MCL, MGDL can still achieve an improvement of about 20% of R on average in localization accuracy over MCL, and an improvement of about 28% of R over ELA. Here we again note that the results from MCL and ELA are based on using about 10% nodes as anchor nodes, while our algorithm does not depend on anchor nodes when computing the global map for the networks.

Figure 5.13 shows the impact of node density over all 60 placements on MGDL and MCL. MGDL is far beyond MCL in different node densities.

5.3.6 Localization Coverage

In comparison with MCL, another advantage MGDL is higher percentage of nodes that could be localized under same node density. The percentage of localized nodes after MCL is around 92% on average, with results at particular values of R ranging from 96% ($2V_{avg}$

= $0.45R$) to 86% ($2V_{avg}$ = $1R$) down to 45% ($2V_{avg}$ = $2R$) [4]. This comes from the fact that MCL is not able to draw enough good samples from an area in which there is overlap of the anchors' radio range. This occurs in general with high maximum node speeds. In the worst case, the new sample set remains empty leading to a non-localized node. The average coverage for MGDL is quite invariant to the speed of the node, but much related to the number of neighbors a node has. Even in our worst simulation cases, the percentage of nodes that could be localized is more than 97%, with the average percentage is 99% for the whole data set in simulation.

5.4 Summary

In this chapter, we proposed an anchor-free localization for Mobile WSNs. By making use of a standard device accelerometer, we proposed a set of movement detection algorithms, and by testing them on Moteiv's Invent motes, we verify that our approach is reasonable. Then based on such movement detection, we provided the whole AF localization algorithm called Mixed Geographic Distributed Localization (MGDL). Based on simulation in NS-2, we found that our algorithm has more accurate localization results than previous mobile localization algorithms. Also, MGDL obtains better coverage than AB mobile localization algorithms that we compared to (MCL). MGDL has flexible communication overhead for both high-mobility and low-mobility nodes, while MCL only has fixed communication overhead for both high-mobility and low-mobility nodes, which may impose overly high communication overhead in the latter case.

Chapter 6

Wormhole Resilient Localization

In this chapter, we study how a localization protocol can be affected by manipulated communication, especially those like wormhole attacks that don't need to capture the keys used in the network, and how such an attack can be detected and defended in distributed scheme. By applying wormhole attack on several localization algorithms, this chapter first shows that wormhole attack is a serious threat to all these algorithms, based on simulation in NS-2. We first evaluate the effects of wormhole attacks on several Anchor-Free localization algorithms based on hop-counting and connectivity techniques. Then, we present an Anchor-Free localization algorithm called Wormhole-resilient Geographic Distributed Localization (WGDL), which embeds a wormhole detecting/recovering mechanism. if this mechanism detects wormhole attacks in location computation, it will restart the localization after freezing the wormhole affected area. Simulations show that the proposed detection method is effective on different network placements, and that the mechanism has both a low False Toleration Rate (FTR) and a low False Detection Rate (FDR) in detecting wormhole attacks.

6.1 Introduction

Currently, most localization algorithms assume that Wireless Sensor Networks (WSNs) are deployed in a trusted environment, but it is possible that a WSN is to be deployed in untrusted environments. In this situation, an adversary can interrupt the functionality of localization algorithms by exploiting vulnerabilities in the localization scheme used. Though many localization techniques have been proposed for wireless ad hoc networks [10, 19, 82, 83], little research has been presented on securing a localization scheme against threats.

There are many different types of attacks against WSNs, such as wormhole attacks, Sybil attacks [61], jamming and packet injection attacks [99] against WSNs. Among them, wormhole attacks [18, 34], which attack a network by manipulating communication, are a threat that does not require knowing the cryptographic infrastructure of a network. This chapter focuses on wormhole attacks against localization schemes. A wormhole is an adversarial topological feature which disturbs the spatiotemporal relationship by manipulating the multi-hop communication in WSNs. In a WSN, usually a wormhole is constructed with some fast connection from one point to another distance point by some hostile adversary. It copies a message from one point quickly, to a distance point.

We use the term Anchor Free (AF) to refer to the methods that use no specially designated reference nodes with known physical coordinates. The alternate to AF localization is Anchor-Based (AB) localization, which relies on some special nodes that already know their exact physical position. Several papers, such as ones on LAD [22], SerLoc [51], and paper [33], have shown that the impact of a wormhole attack is serious in AB localization, since if a wormhole tunnel seriously affects anchor nodes, then the performance of localization will be significantly lower. There are some existing secure localization schemes that have been proposed by using AB methods, which focus on defending anchor nodes

against wormhole attacks, but there has been little research done on wormhole attacks in AF schemes, where there are no anchors to target.

Another classification of localization independent of the AB /AF designation is based on whether distance measurements are used. We use the term Range Free (RF) to refer to localization methods that do not using distance measurements [29, 60, 63, 83]. In comparison with Range-Based (RB) localizations, which use distance or angle measurement to do localization, RF localization usually leads to less costly implementations of sensor nodes. Obviously, RF/RB terms are mostly related to hardware implementation for localization, which have few effects on localization schemes themselves. Here we focus attention on the wormhole attacks in AF localization systems.

In this chapter, we make the following contributions: (i.) we evaluate the impact of wormhole attacks on AF localization schemes; and (ii.) we propose a wormhole-resilient localization in detail. Our algorithm embeds a wormhole detection/recovery mechanism, which can detect wormhole attacks, and restore the localization by freezing the wormhole affected area. We provide extensive simulation for (i) and (ii) in NS-2, which shows that our methods are effective at detecting and defending against wormhole attacks.

The remainder of the chapter is organized as follows. Section 6.2 evaluates the impact of wormhole attacks on localization. Section 6.3 discusses details of our wormhole-resilient localization algorithm. Section 6.4 reports the result of our simulations. And finally, Section 6.5 gives our conclusions.

6.2 Impact of Wormhole Attacks

In a typical wormhole attack, an attacker receives packets at one point in the network, forwards them through a wireless or wired link (referred as "tunnel") with much lower latency than the regular network links and relays those packets at another position in the

network. In this chapter, we assume that a wormhole is bi-directional and comes with two endpoints (referred as "ends"), although in theory multi-end wormholes are possible.

We also assume that each wormhole in a network is (1) passive, and thus does not send out any message without any inbound message, and (2) static, which means that it will not move around.

Based on different measurement techniques, recently proposed Anchor-Free localization algorithms can be categorized into two broad types: one uses connectivity-based localization, and the other uses hop-counting. In the first category, we will use MDS-MAP [83] and MDS-MAP(P) [82] as representative algorithms, since they have been shown to outperform previous localization algorithms based only on connectivity [19, 62]. In the second category, we will use hop-coordinates-based (Chapter 3) GDL localization (Chapter 4), which is a variation on hop-counting with higher accuracy in localization.

6.2.1 Attack Experiments

We implemented three different AF algorithms (MDS-MAP, MDS-MAP(P) and GDL) as routing agents and the bootstrap node for the hop coordinates technique as a protocol agent in NS-2 version 2.29 [57] with 802.15.4 MAC layer [106] and CMU wireless extensions [28]. The configuration parameters used for NS-2 are RF range = 15 m, propagation = TwoRayGround, and antenna = OmniAntenna. Then, we implemented a wormhole as a wired connection with higher throughout to achieve lower latency that forwards packets from one node to another node.

In our experiments, we used uniform placement—n nodes are placed on a grid with r uniform randomized placement error, where r is the width of a small square in the grid. We constructed a total of 30 placements with n = 100, 400, 900, 1600, and 2500, and with r =2, 4, 5, 8, 10, and 12 m, respectively.

We simulated different AF localization algorithms under two configurations of wormholes: (1) only one wormhole with two ends no matter how many nodes there are in total in the network, and (2) a variable number of wormholes (in our experiments we using 5% of the total number of nodes in the network).

6.2.2 Experiment Results

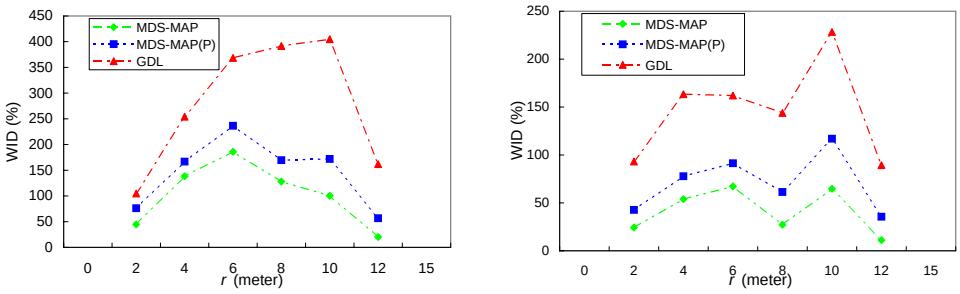
Before we explain the impacts of wormhole attacks, we first give a definition that will enable us to evaluate the impact caused solely by a wormhole attack in the localization. In order to measure the extent that localization error increases when a wormhole is present, we define Wormhole Induced Distortion (WID) by measuring the error with and without the wormhole present:

Wormhole Induced Distortion (WID): evaluates how much localization is affected by the wormhole attack. It is calculated as follows based on localization error (in the formula, shortened as "error"):

$$\text{WID} = \frac{\text{error with wormhole(s) - normal error}}{\text{normal error}} \times 100\% \tag{6.1}$$

Here localization error is computed as the average of the absolute value of the difference between the physical location and the computed location of each node in the network. Suppose that there is a network with a set of nodes A, that the physical location for each node is $\{(x, y)_a | a \in A\}$, and that the computed location for each node is $\{(\grave{x}, \grave{y})_a | a \in A\}$; then localization error $= \frac{\sum_{a \in A} \sqrt{(x-\grave{x})^2+(y-\grave{y})^2}}{|A|}$.

Figure 6.1(a) shows the relationship between WID and the r parameter when the number of wormholes is equal to 5% of the total number of nodes in the network. The figure shows that wormholes affect localization results for all three AF algorithms under the r parameter from $4 - 10$ m, since when $r = 2$ and 12m, the result of localization accuracy,



(a) WID with 5% wormholes (b) WID with one wormhole

Figure 6.1: WID when wormhole(s) exists in WSN

which also acts as the denominator in the equation of WID (Equation 6.1), is not as good as when $4 = 4 - 10$m, so it affects that the effection of wormhole, which is measured with WID, is not as big as when $r = 2, 12$m.

Figure 6.1(b) shows the relationship between WID and one wormhole with two ends under different r values (r = 2,4,6, 8,10,12m). Because there is only one wormhole inside of the network, so the impact of the wormhole is decreased, in comparison with Figure 6.1(a), but the impact of the wormhole still affects the whole range of r in all three algorithms.

It is worth noting that both the single wormhole and 5% wormhole cases affect hop-counting based GDL more than the MDS algorithms; this is easy to understand, since in a hop-counting technique, nodes count the number of hops based on the previous node, and if a wormhole introduces a short it will affect many more nodes in the hop-counting based localization would be the case in connectivity based localization.

So, from these results we can see that the impact caused by a wormhole is serious not only to connectivity-based localization, but also in hop-counting-based localization, even when there is only one wormhole in the network.

6.3 Wormhole-resilient Geographic Distributed Localization (WGDL)

Our algorithm combines wormhole detection/defense into the localization: first, we will introduce the measurement/ probe procedure; then, we will do local map computation; and at last, a wormhole detection/recovery mechanism will be addressed before the transformation procedure, which transforms local maps into a global map. The overview of this WGDL algorithm can be seen in Procedure 16.

Procedure 16 WGDL Algorithm

1: Measurement/Probe Procedure
2: Local Map Computation Procedure
3: **if** detect wormhole attacks **then**
4: First, disable the part of the networks that detects wormhole,
5: then restart the whole localization.
6: **end if**
7: Transformation Procedure to transform the local maps into a global map.

6.3.1 Measurement/Probe Procedure

In the measurement/probe step, we use the hop-coordinates technique (introduced in Chapter 3), which is similar to hop-counting but has more accurate measurement, to flood a message to the network to finish the measurement. A wormhole attack is passive, which means such an attack can only happen when there is some message being transmitted near the wormhole area. So, the messages created and forwarded by the measurement procedure can also used as a probe message to detect whether there are wormhole attacks inside the network. The basic idea is:

(i) In bootstrap node: A bootstrap node (x) creates a measurement/probe message, which includes a variable $hop = 0$, with ($i = x$) to flood the network. After that, the

bootstrap node will drop any measurement/probe message originated by itself.

(ii) In all other nodes in the WSN: Suppose that a node a is calculating its hop distance, and node b is one of the neighboring nodes of node a. After finishes calculating its hop distance, node a forwards that measurement/probe message with $hop = hop + 1$ to its neighboring nodes. Its neighboring nodes will continue blood the message.

The detail of measurement/probe procedures can be found in Section 3.4.2. The total cost for this step is as follows: computational cost of $O(1)$ per node, communication cost of $O(|N_a|)$ per node, memory cost of $O(|N_a|)$ for each node, and of $O(n)$ for the whole network.

6.3.2 Local Map Computation

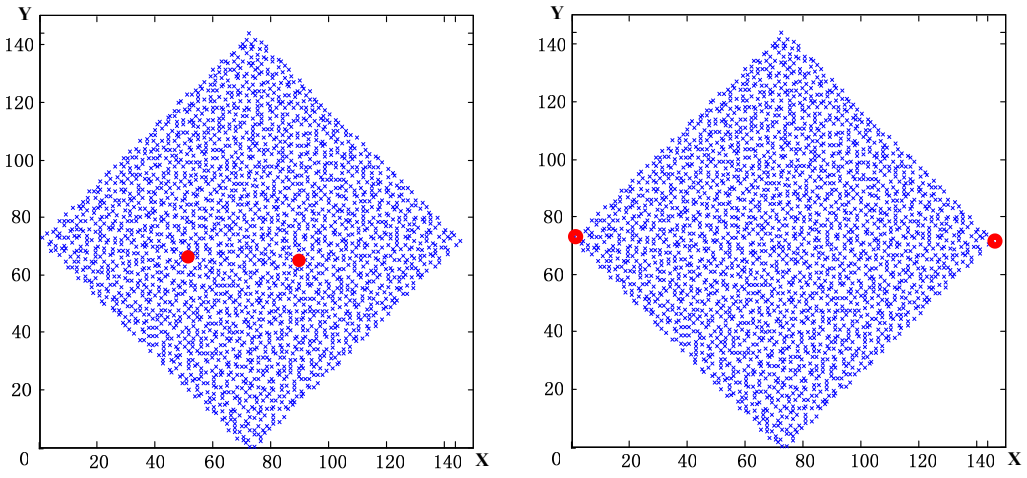
After running the measurement/probe procedure over the whole network, we will consider how to detect wormhole attacks as we continue to do the localization.

In this step, each node will compute a local map for its neighbors based on the hop-coordinate computed in the previous step. After the generation of hop-coordinates in Section 6.3.1,each node will send a request to its neighbor nodes that are within k hops to send back their hop coordinate from some bootstrap node.

After each node receives the hop coordinate from its neighbors, that node will compute shortest paths between all pairs of nodes k hops to that node, using Dijkstra's algorithm or other similar algorithms.

Then, we apply MDS to the $(|N_a|+1)\times(|N_a|+1)$ shortest path matrix (here $|N_a|$ is the number of nodes that can be reached by node A in k hops) and retain the first two (or three) largest eigenvalues and eigenvectors to construct a 2-D local map.

The total cost for this step is a computational cost of $O(|N_a|^3 n)$ and a memory cost of $O(|N_a|^2)$ per node, with $O(1)$ communication cost per node in this step.



(a) A 2500-node WSN with one wormhole

(b) The same WSN with one wormhole sitting on the edges

Figure 6.2: A 2500-node WSN ($r = 2$m) with one wormhole

6.3.3 Attack Detection and Defense

Based on the local map from the previous step, we now turn to the problem of detecting. First, we examine the effects of wormhole attacks on computed local maps.

Observation of a Wormhole in Different Location

In order to observe a wormhole in localization without wormhole detection/defense mechanism, we implemented the measurement/probe procedures (in Section 6.3.1) and the local map computation procedure as routing agents and the bootstrap node for the probe procedures as a protocol agent in NS-2 as same configuration as in Section 6.2.

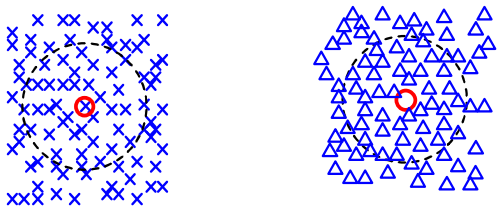
In our first experiment, we used 2500 nodes in a uniform placement— total 2500 nodes are placed on a grid with r uniform randomized placement error, where $r = 2$m is the width of a small square in the grid. A wormhole is implemented as a wired connection.

Figure 6.2(a) and 6.2(b) shows the same sensor network; each 'x' represents a node, and the bold circles (also in red) indicate the two ends of a wormhole; in Figure 6.2(a), the wormhole is sitting in the center of the network, while in Figure 6.2(b), the wormhole is

sitting on the edges of the network.

Since we aim to detect wormholes using a distributed approach and since each WSN node has limited resources to store global information, each node can only use local information to detect wormhole attack.

A Feature to Detect Wormhole Attacks



(a) (b)

Figure 6.3: Two parts of the networks near wormhole ends.The two parts are cut from Figure 6.2(a). Left part of the figure (labeled as (a)) shows the circle, which represents one end of the wormhole, overlaps with a node, which is represented as "x". Right part of the figure (labeled as (b)) shows the circle shows there are several nodes, which are represented as triangle, close to the circle, which represents one end of the wormhole. Here, circles are the wormhole ends, dashed circles means the range of transmission ($R = 15$m here).

Consider the parts of the network with a wormhole with two ends again in Figure 6.3, by selecting two parts of the network that are close to the ends of the wormhole in Figure 6.2(a). We use a dashed circle to represent the neighbor area where a particular node can directly reach in transmission range R, since there are two ends, we shows two parts of the network. Then, after the node, which is overlapped with one end of the wormhole as shown in Figure 6.3(a), finished local map computation for the nodes in its local range, it will get a local map as in Figure 6.4, if we consider the effect of wormhole. From Figure 6.4, we can see that because wormhole shortcuts the two parts of the network, the node, which overlapped with one end of wormhole in Figure 6.3(a), can reach more range than before (if we measure the longest distance in this local map, it will equal 49m.).

To compare with Figure 6.4, we also gave the local map for the same node but without

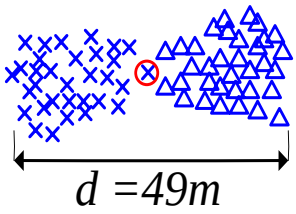


Figure 6.4: Local map is computed after the measurement/probe and local map computation in the node that is overlapped with one end of wormhole in Figure 6.3(a).

considering of the effect of wormhole in Figure 6.5. From this figure, we can see if we measure the longest distance in this local map, it equals around 28m, which is close to $2R = 30$m.

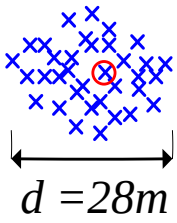


Figure 6.5: Local map without wormhole effect. This Local map is computed after the measurement/probe and local map computation in the node that is overlapped with one end of wormhole in Figure 6.3(a).

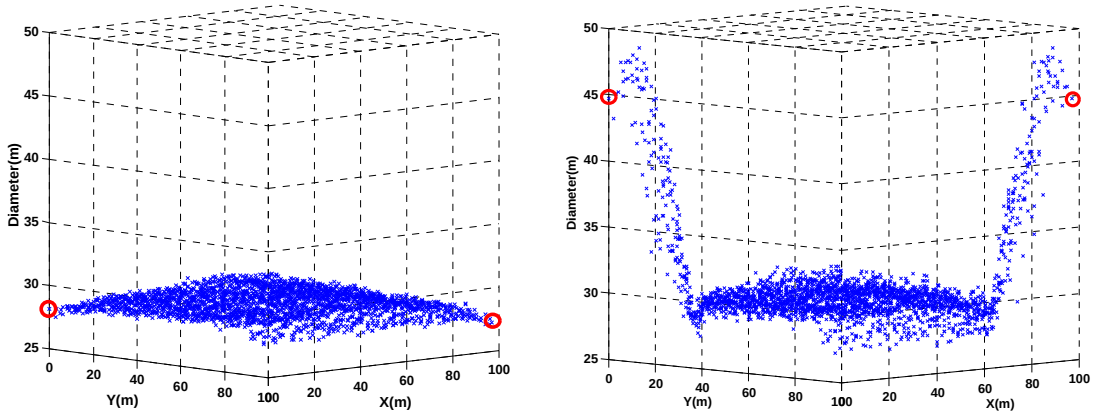
Left part shows the circle, which represents one end of the wormhole, overlaps with a node, which is represented as "X".

From the above observation, we instead focus on detecting wormholes by using a different feature—the diameter of the computed local map. We define diameter d for Node a here:

$$Diameter : d_a = max(distance(i, j)) \tag{6.2}$$

Here $i, j \in N_a$, which is the set of neighbor nodes of Node a.

Theoretically, the diameter of the neighbor area for a node will be approximately double of its transmission range R, since one node only can hear from its neighbors within the



(a)without Wormhole (b)with a Wormhole

Figure 6.6: Diameter measurement without and with wormhole in a 2500-node WSN. The diameter of a local map will roughly be $2 * R$ (from the measurement, the diameter varies from 28 to 36 m, while $R = 15$ m in this figure) unless there is a wormhole attack, in which case the diameter of a local map will become longer as the position draws closer to the wormhole.

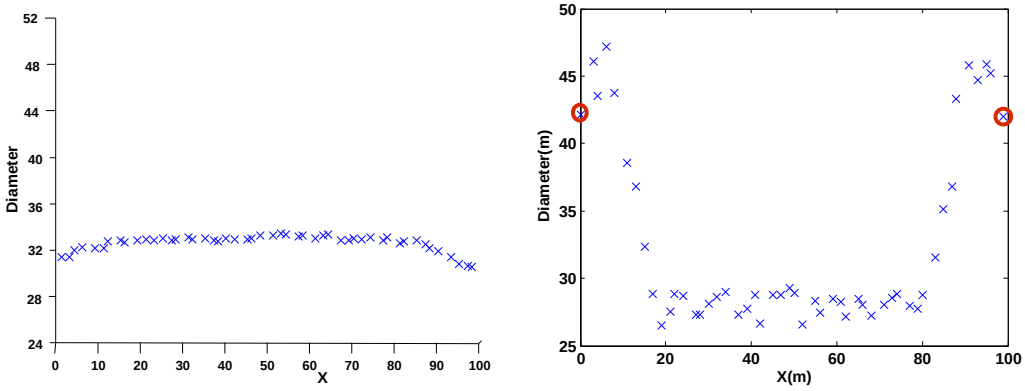
transmission range R. But because of the shortcut of wormhole, the computed map for that neighbor area of that node will be distorted, and so the diameter of that computed local map will be larger than the physical one, as shown in Figure 6.4, we can see $d = 49m$.

In order to verify whether such diameter feature is working in detecting wormhole in the whole network, we compute the diameter for each node in the same 2500-node network with and without wormhole. The results are shown in Figure 6.6(a), if we examine nodes that are very near to a wormhole, such as the area near the circles, which represents the ends of a wormhole, in Figure 6.6(b), the diameters of the local maps for these nodes will be noticeably increased by proximity to the wormhole, comparing the diameters in the same nodes in the network without wormhole in Figure 6.6(a). But if the nodes are a little farther away, or in a distant part of the network, such as the middle area in Figure 6.6(b) , the diameters of the local maps for these nodes, will be almost as normal as these in the same area in Figure 6.6(a), which is without wormhole.

In Figure 6.6(b), the diameter of a local map will roughly vary from 28m to 36m, which

is close to $2*R$, unless there is a wormhole attack, in which case the diameter of a local map will become longer as the position draws closer and closer to the wormhole. The diameter reaches the highest (about 50 m) at the nodes at about 7 m to the ends of wormhole. Then the diameter decreases, the reason of decreasing is because the nodes are approaching to the edges of the network, the nodes can reach less neighboring nodes because of the edges, but the diameter is still above 26 m.

The diameter feature is also good at detect wormhole attack in networks with irregular shapes, and in networks with multiple wormholes inside them. We did some experiments of diameter feature in a network with string topology, and a network with two wormholes inside it.



(a) Diameter measurement without wormhole

(b)Diameter measurement with a wormhole

Figure 6.7: Diameter measurement in a 50-node WSN in string placement without/with a wormhole

In a string topology experiment, we tested a 50-node network, inside of which, each node are uniformly distributed in a 100 meter string in one dimension. First we measure the diameter for each node without any wormhole inside of the network, the result is in Figure 6.7(a). The diameter is at most 32.5 m in Figure 6.7(a). Then, we add a wormhole into the network with the two ends of that wormhole at the two ends of the string. We can see that right now, the diameters of the nodes that are close to the ends of the wormhole are

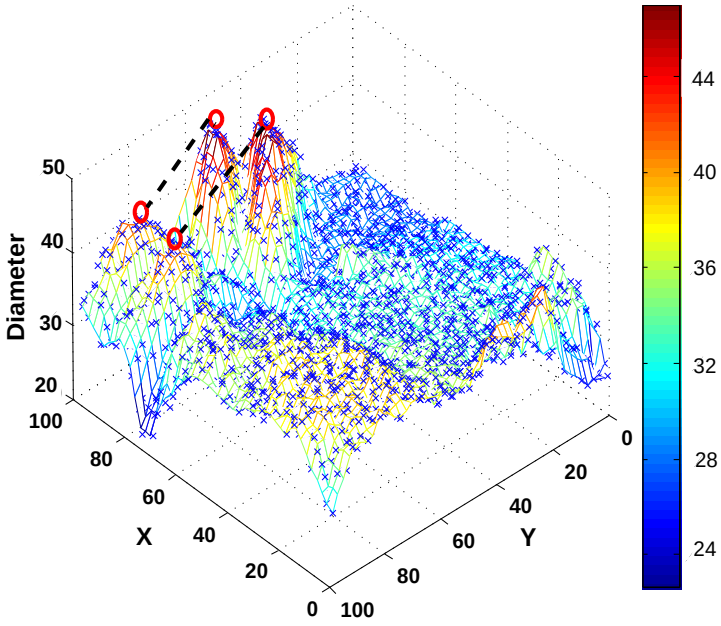


Figure 6.8: Diameter measurement in the 2500-node WSN in Figure 6.2(b) with two wormholes. Here, circles are the ends of wormholes (also in red), dashed lines are the tunnels of the wormholes. An 'X' is represented as a node. The 50X50 mesh is only for visualization purpose. Color bar represents the value of diameter.

larger than 44 m, shown in Figure 6.7(b).

In order to test the feature of 'diameter' in detecting multiple wormholes in a network, we deployed two wormholes in the network of Figure 6.2.a. The measurement of diameter for all nodes as shown in Figure 6.8. The locations of the ends of these two wormholes are represented as circles in the same figure. From the figure, we can see that even two wormholes are very close to each other, the peaks of diameter are still appeared in the nodes that are close to the ends of the wormholes, from our measurement, four peak values are 49.6, 50.4, 44.4, 45.2 m respectively.

So, by computing the diameter d for local map, this detection algorithm can runs independently in each node, in conjunction with the computation of a local map for the neighboring area. Since all nodes in this area are within k hops of the calculating node, the detection algorithm can compute the diameter of each local map after determining each

neighbor node's location.

Detection Procedure

Thus, we propose to use the diameter to determine whether there is a wormhole attack present or not. From the experiment in Figure 6.6, we can see that usually the diameters for local maps will be around $2R$, but if there is a wormhole inside of the network, then the diameters of the local maps, which are computed by the nodes close to the ends of the wormhole, will be much higher than $2R$. So, we can define a threshold for the diameter to detect wormholes inside of the network. Since, the lower the value we assign to such threshold, the higher possibility it is that nodes send the error alarms of wormhole. In order to adjust the sensitivity of detection procedure, we introduce a constant parameter λ. We define a threshold as λR, where $\lambda > 2$, to determine whether there is a wormhole attack present or not.

Suppose the diameter of a local relative map is d; if $d > \lambda R$, where $\lambda > 2$, then we can say there is a wormhole in the network. And if not, we can say that the error probably comes from localization error. The details of the detection algorithm follow.

Suppose node a is an arbitrary node in the WSN. First, we propose a distributed detection Procedure 17, which is used to compute the diameter after running the measurement/probe procedure and local map computation in Section 6.3.1 and 6.3.2.

The total cost for this step is a computational cost of $O(|N_a|^2)$ and a memory cost of $O(|N_a|)$ per node, with no communication cost in this step (if no "FOUND WORMHOLE" message is sent to sink node).

Defense Procedure

From the above observation, it is safe to see that wormhole attacks affect in main the immediate area for localization algorithms. This locality of effect makes it possible to

Procedure 17 Wormhole Detection in node a

1: Input: local map G in node a for $N_a \cup \{a\}$
2: diameter $d_a = 0$
3: **for** each $b \in N_a \cup \{a\}$ **do**
4: **for** each node $c \in N_a \cup \{a\} - \{b\}$ **do**
5: **if** $d < d.est_{(b,c)}$ in local map G **then**
6: $d_a = d.est_{(b,c)}$ in local map G
7: **end if**
8: **end for**
9: **end for**
10: **if** $d > \lambda R$, here $\lambda > 2$ **then**
11: Send FOUND WORMHOLE message to sink node.
12: **end if**

generate an alert regarding wormhole attacks in a region, but since wormhole attacks do not compromise individual nodes, it is not trivial to localize the wormhole precisely or defend against them at the level of individual nodes. Fortunately, if the wormhole detection Procedure 17 detects the wormhole attack, then this can be taken to mean that there is a wormhole end nearby. Thus, in order to defend against wormholes, we propose the idea of "freezing" nodes that have detected wormhole attacks in their vicinity, along with their neighbor nodes, to isolate and negate the effect of a wormhole.

A "freezing" action is defined as a special function that disables a node's communication function. If a node is frozen, then this node will never communicate with other nodes, in other word, the frozen node is disconnected from the network.

Suppose that the wireless range for a wormhole attack equals k times the wireless range R of a normal node; if this is the case, then it is possible that we can stop the transmission of a wormhole attack by freezing the nodes within k times wireless range R of one detecting location.

From a node (or nodes) that detects a wormhole attack, a special message will flood out to freeze neighboring nodes. If the bootstrap node receives this message, it will restart the localization procedure. If other nodes besides bootstrap node receive this message, they

will reset the value of its hop-coordinates to MAX-hop, and reset the status of a node as same as before localization.

For the whole network, this procedure will need $O(1)$ communication cost per node. The advantage of this freezing method is that it can let other parts of the network continue to function, and deals implicitly with multiple wormholes. A disadvantage is that it disconnects some parts of the network to counteract the wormhole.

6.3.4 Transformation Procedure

If the previous step in Section 6.3.3 did not detect a wormhole and restart the localization, again, then in this step, we will continue the localization process, to assemble the local maps that are computed and stored in each node into a global map by calculating the transformation. The basic idea is as same as in Chapter 5 Section 5.2.4.

First, we bootstrap from a pre-assigned node to compute the transformation matrix for that node and follow it by broadcasting out a transformation message to its neighbors to let them start to compute the transformation matrix with their neighbors, too.

Then, after we find the set of neighboring nodes, we compute a transformation matrix for each node b in the neighbor node set N_a of node a.

Suppose that there are two sets of neighbor nodes, N_a and N_b, that are within k hops of the nodes a and b, respectively. Their intersection is $I = N_a \cap N_b$, and we use matrix $I_a = \left\{\ldots,(x_i,y_i)',\ldots\right\}'$ (here (x_i,y_i) are the coordinates of one node i) to represent the coordinates of nodes in the I generated by node a, and similarly $I_b = \left\{\ldots,(\grave{x}_i,\grave{y}_i)',\ldots\right\}'$ for node b. We can then compute a transformation matrix T such that it minimizes $\sum\sqrt{((x_i-\check{x}_i)^2+(y_i-\check{y}_i)^2)}$, where $(x_i,y_i)\in I_a$, $(\grave{x}_i,\grave{y}_i)\in I_b$, and $(\check{x}_i,\check{y}_i)\in I_b\times T$. The procedure is given in Procedure 18.

If a node receives a transformation matrix *T*, then it will first check whether it is already

Procedure 18 Compute transformation matrix *T* for N_a in node a

1: set *self* as "transformed"
2: **for** each node $b \in N_a$ **do**
3: request N_b from node *b*
4: $I = N_b \bigcap N_a$
5: generate I_b and I_a from I
6: compute transformation matrix T_b such that $\sum \sqrt{((x_i - \check{x}_i)^2 + (y_i - \check{y}_i)^2)}$ is minimized, here $(x_i, y_i) \in I_a$, $(\check{x}_i, \check{y}_i) \in I_b \times T_b$.
7: send matrix T_b to node *b*
8: **end for**

Procedure 19 Receive transformation matrix T in node a

1: INPUT: matrix T_a from some neighboring node
2: **if** this node is transformed **then**
3: drop (message T_a)
4: RETURN
5: **end if**
6: transform the local map F_a with T into a portion of global map H
7: send back corresponding coordinates in the local map to the neighboring nodes in N_a
8: set *self* as "transformed"
9: **for** each node $b \in N_a$ **do**
10: request N_b from node *b*
11: $I = N_b \bigcap N_a$
12: generate I_b and I_a from I
13: compute transformation matrix T_a such that $\sum \sqrt{((x_i - \check{x}_i)^2 + (y_i - \check{y}_i)^2)}$ is minimized, where $(x_i, y_i) \in I_a$, $(\check{x}_i, \check{y}_i) \in I_b \times T_a$.
14: send matrix T_b to node *b*
15: **end for**

transformed or not; if so, the node will drop such a message, and if not, it will apply procedure 19. This will allow the node to compute the local map (which will be a part of the global map), which it will then send to its $|N_a|$ neighbor nodes. At the same time, the node will find its neighbor node set, and will compute transformation matrix T for each node in the set using procedure 19.

After we have flooded the network for the transformation step in the whole networks, we achieve the global map for the whole network, which is stored in a distributed way in each node in the network.

Total cost for this step: computational cost of $O(|N_a|)$, memory cost of $O(|N_a|)$ for nodes, and communication cost of $O(1)$ for each node or $O(n)$ for whole network.

6.4 Simulations and Results

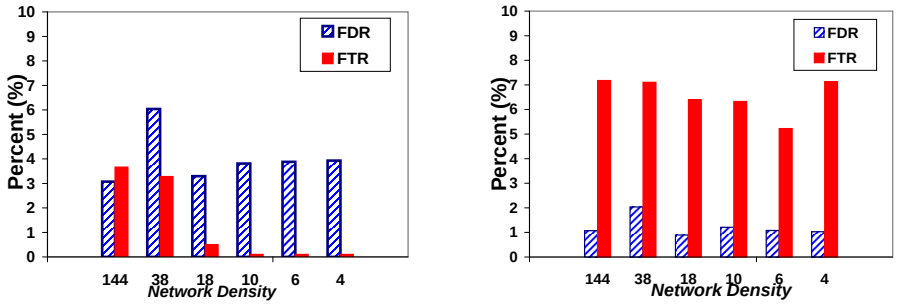
As same as in Section 6.2, we implemented our localization algorithm as a routing agent in NS-2. The placements of the two ends of a wormhole are uniformly randomized inside of the network.

6.4.1 Detection Simulation Results

Metrics

As we decrease the threshold in Procedure 17, we can increase the possibility of detecting wormhole attack, but the possibility of false alarm will be increased. In order to evaluate the accuracy of our wormhole attack detection under different λ values, we introduce the following concepts:

False Detection Rate (FDR): the frequency with which a detection system falsely recognizes the errors, which are not caused by wormholes, as the errors caused by wormholes.



(a) when $\lambda = 2.4$ (b) when $\lambda = 2.8$

Figure 6.9: False Detection Rate (FDR) and False Toleration Rate (FTR) for various node placements

FDR = (number of normal localization errors flagged as detected wormholes while there is no wormhole) / (total number of trials)

False Toleration Rate (FTR): the frequency with which the detection system falsely recognizes the errors caused by wormholes as normal localization error. FTR is used to find how frequent our algorithm is to fail to detect a wormhole while there is a wormhole in the system.

FTR = (number of wormhole attacks are not detected under the condition that there is a wormhole) / (total number of trials).

Detection Simulation Result

In practice, we select two λ values to compare FDR and FTR: $\lambda = 2.4$ and $\lambda = 2.8$. We first evaluate the result of WGDL in detecting whether there is a wormhole inside of the networks. Results in terms of FTR and FDR are shown in Figure 6.9. Our WGDL algorithm has low FDR with FTR=0 in some cases when $\lambda = 2.4$ as in Figure 6.9(a); when $\lambda = 2.8$ as in 6.9(b), our detection algorithm can achieve a low FTR with FDR approaching zero.

6.4.2 Defense Simulation Results

In this section, we will first measure localization errors of WGDL without wormhole attacks by comparing with other localization algorithms such MDS-MAP and MDS-MAP(P), then test WGDL with/without defense mechanism under wormhole attacks in NS-2. We tested the defense procedures with one wormhole attack, which was uniformly randomized and included in every placement. We tested all the placements (# of nodes = 100, 400, 900, 1600, 2500, $r = 2$, 4, 6, 8, 10, 12) under $\lambda = 2.4$ and $\lambda = 2.8$.

Localization Error in Simulation

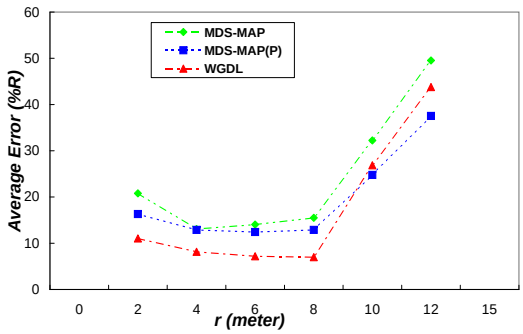
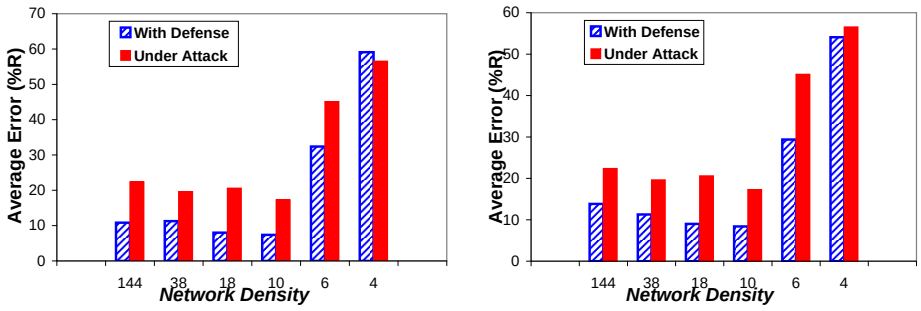


Figure 6.10: Localization error without wormhole in WGDL comparing with MDS-MAP and MDS-MAP(P)

Based on these results, we can compute the overall average accuracy corresponding to the various network densities (corresponds to various grid sizes (r)) tested. The overall results are shown in Figure 6.10. We can see that our algorithm improves the accuracy of localization about 28% and 10% respectively compared with the MDS-MAP and MDS-MAP(P) algorithms under different network densities. And the accuracy of WGDL is close to the accuracy of our GDL algorithm in Chapter 4, where there is no wormhole inside of the network. The reason why there is differece between the curve of WGDL curve in this figure and the curve of GDL in Figure 4.13, is that the results in Figure 6.10 came from

the experiments in the networks where $n = 2500$, while in Figure 4.13, the result of GDL came from the experiments in the network where n from 36 to 2500.

Defense Simulation Result



(a) when $\lambda = 2.4$ (b) when $\lambda = 2.8$

Figure 6.11: Defense procedure performance evaluation. Here, we did not count the accuracy of disabled nodes because of the wormhole attack.

We test WGDL with/without defense mechanism under wormhole attacks in NS-2. We tested the defense procedures with one wormhole attack, which was uniformly randomized and included in all placement described before, under $\lambda = 2.4$ and $\lambda = 2.8$.

Figure 6.11 shows the results when $\lambda = 2.4$ in Figure 6.11(a) and $\lambda = 2.8$ in Figure 6.11(a) under the same condition: one wormhole (with two ends) and the GDL localization algorithm using hop-coordinates technique. From the above figures, we can see that the wormhole-defense mechanism inside of the WGDL can decrease impacts of wormhole attacks by between about 65% (when $\lambda = 2.4$) and 71% (when $\lambda = 2.8$), compared to the case of not considering the defense mechanism. When $\lambda = 2.8$, there is, however, a higher possibility of wrong alarms to wormhole, as shown in Figure 6.9.

From Figure 6.12, we can see if a defense is more aggressive (here $\lambda = 2.8$, comparing $\lambda = 2.4$), the algorithm can reach better results when the network density is lower, but when a defense is not so aggressive (such as when $\lambda = 2.4$), it usually achieves better

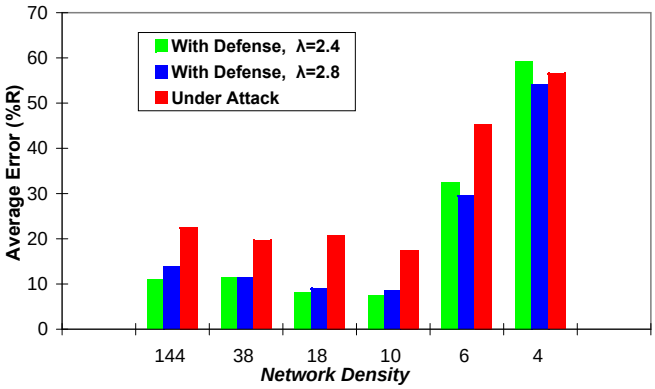


Figure 6.12: Comparison of different λ in defense procedure performance evaluation results when the network density is higher.

6.5 Summary

In this chapter, we evaluate how serious the impact of a wormhole attack can be in AF localization algorithms. Our simulation results show that in the two AF localization approaches we identify, (1) hop-counting-based localization (represented by GDL localization in our simulation), and (2) connectivity-based localization (represented by MDS-MAP & MDS-MAP(P)), the accuracy of localization is impacted considerably.

We proposed a Wormhole-Resilient Anchor-free Localization algorithm called WGDL for WSNs. First, we gave a measurement/probe procedure to measure the distance to some bootstrap node. After we introduce the local map computation procedure, we propose a detection method based on computing 'diameter' in local maps. The results obtained from simulations show that proposed method worked well at detecting wormhole attacks. Moreover we provided effective defense methods to the attacks that experiments showed could be harmful to localization.

Right now, we are basing experiment to decide the threshold and λ in deciding whether a diameter measurement triggers an alarm for wormhole. One future work may need to

improve our algorithm is how to decide such threshold and λ automatically.

In this chapter, though we focus on detection/defense algorithms only for one wormhole attack in WSNs, with examples we shown that the same procedures can also be used to detect wormhole attack in networks with irregular shapes, and in networks with multiple wormholes inside them.

From a node (or nodes), which detects wormhole attack, a special message will flood out to freeze neighboring nodes. If the bootstrap node (x) receives this message, it will restart the wormhole detection algorithm again, while other nodes receive such message will reset the value of hop-coordinate inside itself. Such process will be ended until there is no node detects any wormhole attack.

Chapter 7

Conclusion

In this chapter, we summarize the contributions, limitations and future research for our work.

Wireless sensor networks are an emerging research field with many challenging issues that need to be solved. More geographic-related protocols developing currently are urging context-aware services. And more context-aware applications are developing in large-scale wireless sensor networks currently. Such applications involve healthcare, surveillance, building/bridge monitoring and other applications. Localization is one of the fundamental components for the above context-aware system and application implementations.

A number of research activities have laid the groundwork for localization in static WSNs and mobile WSNs. However, the solutions for localization in static WSNs can not be applied to mixed WSN case without a significant increase in communication and computational costs. Other work in localization for mobile WSNs relies on the assumption that there are a significant number of anchor nodes in the networks. Also, these algorithms are not efficient in communication and computational cost, which is a big weakness for resource-constrained WSNs when they are applied into mixed WSNs.

Consequently, without the help of anchor nodes, none of the proposed systems can

provide such a localization service for mixed WSNs in efficient, accurate and reliable way. As talked in Section 1, such mixed WSNs have a combination of static nodes and mobile nodes. Each of them can vary from static state to mobile state, and vice versa.

In this dissertation, we first defined the concepts of mixed WSNs, the localization problems in mixed WSNs and the reason why we selected the Anchor-Free scheme. Then, we described a series of GDL algorithms to satisfy the requests for the accuracy, efficiency (low communication and computational cost) and reliability (protection from manipulation of communication) of the localization without anchor nodes in a mixed WSN.

While accuracy is an important consideration for localization in a mixed WSN, we developed a new measurement approximation "hop-coordinates", to improve the accuracy of measurement as well as localization.

Efficiency is critical, especially for resource-constrained WSNs. By integrating sensor components into our methods, our GDL algorithm can help to decrease the computational cost as well as communication cost when few nodes are moving or nodes are moving slowly, which are common cases especially in large-scale mixed WSNs.

Reliability against communication manipulation is another issue for localization. In this dissertation, we only focus on the wormhole attack, and we developed a distributed attack detection/defense mechanism embedded into our localization against such attack.

The primary contributions of this dissertation are:

- An accurate measurement technique for anchor-free localization – "hop-coordinates". Compared to other methods in the same category, with the same data, this measurement technique has better accuracy than other methods. The accuracy stands when applying this technique into our localization.

- An anchor-free localization for static WSNs – "GDL". By integrating with the measurement technique we developed, we created a localization algorithm for static

WSNs without anchor nodes. This localization algorithm provides higher accuracy as well as constant memory cost and communication/computational cost when compared to other algorithms.

- An anchor-free localization algorithm is designed to compute location information both in static and mixed WSNs without anchor nodes – "MGDL". The algorithm is efficient in communication and computation cost, and can be fitted into resource-constrained wireless sensor nodes. Our algorithm does not rely on other routing algorithms, as well as anchor nodes.

- A reliable localization algorithm – "WGDL", which can detect the wormhole attack manipulated communication in a network, locate the attack, and recover the localization without relying on any centralized server or specific hardware.

Besides our contributions, we must note that most of our simulation results were generated by NS-2 simulator, and some results came from our experiments within a limited number of Tmote Invent motes in limited time period. So, our conclusions may or may not apply to the real-world applications, while some assumptions in NS-2 (for instance, homogeneous radio range assumption in NS-2) may not stand in the real world experiments. Since we only tested our algorithms in a limited number of network placements, it is possible that the results we presented here are different from the results generated from some placements that are different from the placements used here.

The simulation results presented here may not recur exactly in the real world applications. But these results verify the correction of our algorithm in a qualitative way and help to understand the performance in different placements. Although some of they are hard to do in the real world (for example, right now the largest wireless sensor network testbed only consists of 800 nodes [9], we can easily test our algorithms on more than 800 node networks in simulation).

7.1 Limitations and Discussion

Here we talk about some problems and limitations that were found when we did our research, and discuss some possible considerations when applying our algorithms in real applications.

7.1.1 The Considerations about the Simulator We Used

While the most of our simulation results in this dissertation were coming from simulation environment, we talk about some reasons why we used NS-2 in this dissertation in this section.

While experimentation provides a means for exploring the "real world", simulation is restricted to exploring a constructed, abstracted model of the real world. For instance, NS-2 simulator can not simulate the complexity of the environment in the real world [23]. But, instead of using Matlab as other people did in MDS-MAP papers [83, 82], or using pure Java as other people did for MCL [31], we implemented our algorithms on the top of NS-2 with wireless extension from CMU to simulate the wireless sensor networks.

The reasons we used NS-2 instead of Matlab or Java are because Matlab does not provide sophisticated components, which are necessary for a scalable wireless network simulator. For example, Matlab does not include functions to implement dynamic network placement, antenna models, protocol models, all of which are provided in NS-2 simulator [13]. The simulation results from Matlab without sophisticated models of wireless sensor networks would not be sufficient. As to Java, since it is only a programming language, there is no support for network simulation.

Also, we extended NS-2 to let it fit to WSNs simulation, since IEEE 802.15.4 protocol is a dominated MAC and PHY protocol in current commercial wireless sensor node's implementations, such as Xbow's MicaZ and Moteiv's Invent. We replace the original MAC

protocol and PHY layer in NS-2 with IEEE 802.15.4 PHY/MAC extension [106].

We must note that the reasons given above are by no means to imply that the simulation method we used in this dissertation have the capability to replace real world experiments. We only say that we are trying in this dissertation to make our simulation results more realistic.

7.1.2 Heterogeneous Radio Range R

We assumed that the WSN nodes in this dissertation are homogeneous, which means that these nodes have same radio range R, and share the same hardware implementation with the same size of memory, processor and battery. But in real world applications, it is possible that different nodes have different radio ranges, and this irregularity may have an impact on the accuracy of measurement as well as localization. In order to keep the accuracy of our technique in real world application, we considered some methods to alleviate the impact of radio irregularity.

There are two cases we considered for the radio irregularity. One case is that all nodes in a network share the same radio hardware. Another case is that different nodes have different devices with different radio ranges, one example is heterogeneous WSNs. In this case, the impact of radio irregularity depends on the diversity of radio range. In order to alleviate the impact of radio irregularity, we can calibrate each node before deploying them into the field and modify each node to similar radio range by decreasing or increasing the transmission power level, which is supported in current commercial WSN nodes such as Tmote Invent [39].

7.1.3 Message Dropping

In our implementation of GDL series algorithms, we are assuming that the communication between nodes is reliable in simulation by using acknowledgment messages. But it was already observed that message dropping is a fundamental problem in real WSNs applications. For instance, about 20% to 30% [96] messages are dropped in a survey of several typical WSN applications. In order to avoid message dropping in real world implementation, one way is to include some reliable transport protocol such as PSFQ [96]. We also implemented and tested a simple reliable transport protocol in Tmote Invent nodes.

The detail of this reliable transport protocol is as follows. It is constructed by three different components, procedure for sender and procedure for receiver and a k data message FIFO buffer for sender procedure:

1. A node puts every data message planning to send out into a FIFO buffer

2. A sender procedure tries to read a data message (this data message is still in FIFO buffer right now) from the FIFO buffer. If the buffer is not empty, then it sends the data message read from the buffer out, and waits for an ACK message. If in a small constant time, the sender program receives the ACK message for this data message, then the sender procedure deletes this message from FIFO buffer, otherwise the sender procedure will stop wait, and repeat step 2. (The message with ten failed trials will be deleted from the FIFO buffer.)

3. A receiver procedure running in another node. If this receiver procedure receives a message from air, then it sends out an ACK message back for that data message.

We implemented this simple reliable transport protocol and did our experiment in an eight Tmote Invent notes network with $k = 10$, all of which are in one hop area (all nodes in a ten meter circle), and FIFO outbound message rate is twice than FIFO inbound message rate. When we test the above simple reliable transportation program in fifty message per

second (one message includes 29 bytes) transmission rate, there is no message dropped in a total of 2000 message transmissions. Comparing the experiment on the same network without the reliable transportation program, there are about 22% messages dropped.

So, for message dropping problem, even by using this simple reliable transport protocol, we can still have potential to alleviate it significantly. If there are multiple message types, then we can implement one reliable transport protocol for each message type. One main problem for this simple reliable transportation program is that a receiver procedure can receive some duplicate data message occasionally. This problem is a standard network issue as pointed out by Prof. Kotz, the problem is difficult to be solved. Since it is not the main focus of this dissertation, we will not talk some possible solution for it, while there are already several mature solutions in literature.

7.2 Future Work

In this section, we talk about potential future research work for localization.

7.2.1 Localization in an Environment with Obstacles

Obstacles are always a practical problem for the implementation of localization systems in a real environment. Obstacles in the environment will significantly affect the accuracy of the measurement process in a localization system, as well as the accuracy of localization. One way to alleviate this effect is to develop a measurement technique with better accuracy so as to increase the accuracy of localization. Another way is to combine with different measurement techniques, so that, even if one measurement technique is affected by the obstacles in the environment, another technique can still function normally. For example, ultrasound-based measurement technique will be blocked by physical obstacles, while radio-based localization sensor can still pass through these obstacles if they are not

constructed with metal. So, it is possible that we implement some localization systems by combining different measurement techniques together to improve the accuracy of localization.

7.2.2 Outlier Detection in Localization

"Outlier" nodes in localization are defined as nodes that are NOT localized after the localization procedure is finished. The outlier problem exists such that current localization can not localize all the nodes in a WSN. The Outlier problem has been found as a challenge in the deployment of WSNs [84], especially in the mobile case.

When we did the experiment for both our algorithms and other comparing algorithms in Chapter 5, we found many outlier nodes in anchor-based localizations, though our GDL localization already decreased the number of outlier nodes dramatically compared with other anchor-based methods (see comparison results in Section 5.3.6). The outlier problem is still a concern that may hurt localization. For instance, if such an outlier node is a sink node, it will block all communication from other wireless sensor nodes. Although we do not currently include outlier detection in our GDL algorithms, an outlier detection algorithm would be a benefit. If such outlier detection algorithm can be included in the localization system, then we could consider using some actuator such as a mobile robot with wireless sensor nodes to connect the outlier nodes with the main part of a network.

7.2.3 Improving Accuracy with Multiple Measurement Techniques

As we found in our experiment, the accuracy of the measurement technique dominated the accuracy of the localization. So, an accurate measurement technique is a key to an accurate localization system. One way to improve the accuracy of measurement is to overcome the limitation of current techniques such as the methods we proposed to improve the hop-

counting and connectivity-based techniques. Another way is to find a new measurement technique. One good candidate technique is called Radio Interferometry (RI) [56], which claims a high measurement accuracy without any additional hardware.

The series of GDL algorithms are based on a flexible framework, which is constructed with three components: measurement, computation and transformation. By replacing the old measurement technique in the current GDL algorithms, we can simply make it possible to integrate potential new measurement techniques into our current system.

7.2.4 The Possibility of Applying Other MDS Algorithms in Localization

Besides the classical MDS we used in our GDL algorithms, there are other MDS algorithms that have potential to be applied into localization: weighted MDS and replicated MDS.

The difference between weighted MDS and classical MDS is that weighted MDS assumes the similarity matrix for each object is in systematically nonlinear. In a theoretical environment, usually current measurement techniques can return linear value for distance. But in practice, especially if there are many obstacles in the environment where a WSN is deployed, the similarity matrix will not be keeping linear. In this case, it is valuable that we consider applying weighted MDS into such a complicated environment.

Replicated MDS is another variation of MDS that can be used to deal with multiple similarity matrices. Because in current GDL algorithms, we assume there is only one distance matrix for each node to collect, we did not apply replicated MDS in our GDL algorithms in this dissertation. While the technology is developing, it is possible that future sensor nodes can deploy different measurement devices together. So each node may generate multiple distance matrices from different measurement devices. Replicated MDS will be a good candidate to deal with multiple distance matrices to calculate location for each node.

7.2.5 Improving Energy Efficiency in Localization

Energy cost is another big issue for any implementations of algorithms in WSNs. Our algorithms, especially MGDL, already decreased the communication cost significantly compared with other algorithms. In our future work, we are planning to add a synchronization protocol into localization, so that we can develop some adaptive sleep/wake algorithm for GDL. These protocols are based on the speed of movement. If a node is moving very fast, then such adaptive sleep/wake algorithm can wake up the node very frequently. While such a node is moving very slow or becomes static, then such sleep/wake algorithm can make the node turn into sleep state to save the energy.

Bibliography

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A Survey on Sensor Networks. *Communications Magazine*, 40(8):102–114, 2002.

[2] J. N. Ash and L. C. Potter. Robust system multiangulation using subspace methods. In *Proceedings of the 6th International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 61–68, Cambridge, Massachusetts, USA, 2007. ACM Press.

[3] J.N. Ash and L.C. Potter. Sensor Network Localization via Received Signal Strength Measurements with Directional Antennas. In *Proceedings of the 2004 Allerton Conference on Communication, Control, and Computing*, 2004.

[4] A. Baggio and K. Langendoen. Monte-carlo localization for mobile wireless sensor networks. In *Proceedings of the 2nd International Conference on Mobile Ad-hoc and Sensor Networks (MSN)*, Hong Kong, China, December 2006.

[5] P. Bahl and V.N. Padmanabhan. RADAR: an In-Building RF-based User Location and Tracking System. In *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2000.

[6] F. Bai and A. Helmy. A Survey of Mobility Modeling and Analysis in Wireless Ad hoc Networks. *Wireless Ad hoc and Sensor Networks*, 2004.

[7] F. Bai, N. Sadagopan, and A. Helmy. IMPORTANT: a framework to systematically analyze the Impact of Mobility on Performance of Routing Protocols for Adhoc Networks. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 2, 2003.

[8] A. Basu, J. Gao, J. S. B. Mitchell, and G. Sabhnani. Distributed localization using noisy distance and angle information. In *Proceedings of the 7th ACM International Symposium on Mobile Ad-hoc Networking and Computing (MobiHoc)*, pages 262–273, Florence, Italy, 2006. ACM Press.

[9] UC berkeley. 800 Node Self-organized Wireless Sensor Network. In *URL http://webs.cs.berkeley.edu/800demo/*.

[10] R. Bischoff and R. Wattenhofer. Analyzing Connectivity-based Multi-hop Ad-hoc Positioning. In *Proceedings of the 2nd IEEE Annual Conference on Pervasive Computing and Communications (PerCom)*, pages 165–174, 2004.

[11] I. Borg and P.J.F. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer, 2005.

[12] P. Bose, P. Morin, I. Stojmenović, and J. Urrutia. Routing with Guaranteed Delivery in Ad Hoc Wireless Networks. *Wireless Networks*, 7(6):609–616, 2001.

[13] L.E. Breslau, D. Fall, K. Floyd, S. Heidemann, J. Helmy, A. Huang, P. McCanne, S. Varadhan, and K.Y.X.H. Yu. Advances in network simulation. *Computer*, 33(5):59–67, 2000.

[14] J. Bruck, J. Gao, and A.A. Jiang. Localization and Routing in Sensor Networks by Local Angle Information. In *Proceedings of the 6th ACM International Symposium*

on Mobile Ad-hoc Networking and Computing (MobiHoc), pages 181–192, Hong Kong SAR, China, May 2005. ACM Press.

[15] N. Bulusu, J. Heidemann, and D. Estrin. GPS-less Low-Cost Outdoor Localization for Very Small Devices. *IEEE Personal Communications Magazine*, 7(5):28–34, October 2000.

[16] W. Burgard, D. Fox, D. Hennig, and T. Schmidt. Estimating the Absolute Position of a Mobile Robot Using Position Probability Grids. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 896–901, 1996.

[17] T. Camp, J. Boleng, and V. Davies. A Survey of Mobility Models for Ad Hoc Network Research. *Wireless Communications and Mobile Computing*, 2(5):483–502, 2002.

[18] S. Čapkun, L. Buttyán, and J.P. Hubaux. SECTOR: Secure Tracking of Node Encounters in Multi-hop Wireless Networks. In *Proceedings of the 1st ACM workshop on Security of AD-HOC and Sensor Networks*, pages 21–32. ACM Press, 2003.

[19] S. Čapkun, M. Hamdi, and J.P. Hubaux. GPS-free Positioning in Mobile Ad Hoc Networks. *Cluster Computing*, 5(2):157–167, 2002.

[20] T.F. Cox and M.A.A. Cox. *Multidimensional Scaling*. Chapman & Hall/CRC, 2001.

[21] K. Dantu, M. Rahimi, H. Shah, S. Babel, A. Dhariwal, and G.S. Sukhatme. Robomote: Enabling Mobility in Sensor Networks. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN)*, Los Angeles, California, USA, April 2005. IEEE Press.

[22] W. Du, L. Fang, and N. Peng. LAD: Localization Anomaly Detection For Wireless Sensor Networks. *Journal of Parallel and Distributed Computing*, 66(7):874–886, 2006.

[23] S. Floyd and V. Paxson. Difficulties in simulating the Internet. *IEEE/ACM Transactions on Networking*, 9(4):392–403, 2001.

[24] S. Forrest, S.A. Hofmeyr, A. Somayaji, and T.A. Longstaff. A Sense of Self for Unix Processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, volume 120, 1996.

[25] S. Forrest, A.S Perelson, L. Allen, and R. Cherukuri. Self Nonself Discrimination in a Computer. In *Proceedings of the 1994 IEEE Symposium on Security and Privacy*, pages 202–212, 1994.

[26] L. Girod, V. Bychkovskiy, J. Elson, and D. Estrin. Locating tiny sensors in time and space: A case study. In *Proceedings of the International Conference on Computer Design (ICCD)*, 2002.

[27] Chipcon Group. CC2420 Datasheet, 2005.

[28] The CMU MONARCH Group. Wireless and Mobility Extensions to ns-2. In *Obtain via http://www.monarch.cs.cmu.edu/cmu-ns.html*.

[29] T. He, C. Huang, B.M. Blum, J.A. Stankovic, and T. Abdelzaher. Range-free Localization Schemes for Large Scale Sensor Networks. In *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 81–95, San Diego, California, USA, September 2003. ACM Press.

[30] B. Hoffmann-Wellenhof, J. Collins, and H. Lichtenegger. *Global Positioning System: Theory and Practice*. Springer-Verlag, 1993.

[31] L. Hu and D. Evans. Localization for Mobile Sensor Networks. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 45–57, Philadelphia, Pennsylvania, USA, October 2004. ACM Press.

[32] L. Hu and D. Evans. Using Directional Antennas to Prevent Wormhole Attacks. In *Proceedings of the 11th Network and Distributed System Security Symposium (NDSS)*, pages 131–141, San Diego, California, USA, February 2004.

[33] Y.C. Hu, A. Perrig, and D.B. Johnson. Wormhole Detection in Wireless Ad Hoc Networks. Technical Report Tech. Rep. TR01-384, Department of Computer Science, Rice University, June 2002.

[34] Y.C. Hu, A. Perrig, and D.B. Johnson. Packet Leashes: A Defense against Wormhole Attacks in Wireless Ad Hoc Networks. In *Proceedings of 22th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, April 2003.

[35] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden. CarTel: a Distributed Mobile Sensor Computing System. In *Proceedings of the 4th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 125–138. ACM Press, 2006.

[36] Hervé Brönnimann Alex Delis Hüseyin Akcan, Vassil Kriakov. GPSFree Node Localization in Mobile Wireless Sensor Networks. In *Proceedings of the 5th ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE)*, Chicago, Illinois, USA), June 2006.

[37] Crossbow Technology Inc. MICAz Datasheet, 2006.

[38] Freescale Inc. MMA7260Q 3D Accelerometer. In *URL http://www.freescale.com/*.

[39] Moteiv Inc. Invent Motes. In *URL http://www.moteiv.com*.

[40] XBow Inc. TelsoB Motes. In *URL http://www.xbow.com*.

[41] H.S. Javitz and A. Valdes. The NIDES Statistical Component: Description and Justification. Technical Report SRI Annual Report A010, SRI International Computer Science Laboratory, Menlo Park, California,, March 1993.

[42] D.B. Johnson and D.A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353, pages 153–181. Kluwer Academic, 1996.

[43] B. Karp and H.T. Kung. Gpsr: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 243–254. ACM Press, 2000.

[44] L. Kleinrock and J. Silvester. Optimum Transmission Radio for Packet Radio Networks or Why Six is a Magic Number. In *Proceedings of the IEEE National Telecommunications Conference*, volume 4, pages 1–4, 1978.

[45] C. Ko. Logic Induction of Valid Behavior Specifications for Intrusion Detection. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 142–153, 2000.

[46] C.C.W. Ko. *Execution Monitoring of Security-critical Programs in a Distributed System: A Specification-based Approach*. PhD thesis, University of California, Davis, 1996.

[47] J. Kong, Z. Ji, W. Wang, M. Gerla, R. Bagrodia, and B. Bhargava. Low-Cost Attacks against Packet Delivery, Localization and Synchronization Services in Underwater

Sensor Networks. In *Proceedings of the 4th ACM Workshop on Wireless security (WiSe)*, pages 87–96. ACM Press New York, NY, USA, 2005.

[48] D. Kotz, C. Newport, R.S. Gray, J. Liu, Y. Yuan, and C. Elliott. Experimental Evaluation of Wireless Simulation Assumptions. In *Proceedings of the 7th ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, pages 78–82. ACM Press, October 2004.

[49] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric Ad-Hoc Routing: Of Theory and Practice. In *Proceedings of the 21st Annual Symposium on Principles of Distributed Computing (PODC)*, pages 63–72. ACM Press New York, NY, USA, 2003.

[50] T. Lane and C.E. Brodley. Temporal Sequence Learning and Data Reduction for Anomaly Detection. In *ACM Transactions on Information and System Security*, volume 2, pages 295–331. ACM Press New York, NY, USA, 1999.

[51] L. Lazos and R. Poovendran. SeRLoc: Secure Range-Independent Localization for Wireless Sensor Networks. In *Proceedings of the 3rd ACM Workshop on Wireless security (WiSe)*, pages 21–30. ACM Press New York, NY, USA, 2004.

[52] J. Li, J. Jannotti, D.S.J. De Couto, D.R. Karger, and R. Morris. *A Scalable Location Service for Geographic Ad Hoc Routing*. ACM Press, August 2000.

[53] X. Lin and I. Stojmenovic. GEDIR: Loop-Free Location Based Routing in Wireless Networks. In *Proceedings of IASTED Int. Conf. on Parallel and Distributed Computing and Systems (PDCS)*, pages 1025–1028, 1999.

[54] D. Liu, P. Ning, and W.K. Du. Attack-Resistant Location Estimation in Sensor Networks. In *Proceedings of the 4th International Symposium on Information Pro-*

cessing in Sensor Networks (IPSN), pages 99–106, Los Angeles, California, USA, April 2005. IEEE Press.

[55] D. Malan, T. Fulford-Jones, M. Welsh, and S. Moulton. CodeBlue: An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care. In *Proceedings of the MobiSys Workshop on Applications of Mobile Embedded Systems (WAMES)*, 2004.

[56] M. Maróti, P. Völgyesi, S. Dóra, B. Kusỳ, A. Nádas, Á. Lédeczi, G. Balogh, and K. Molnár. Radio Interferometric Geolocation. In *Proceedings of the 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 1–12. ACM Press, 2005.

[57] S. McCanne and S. Floyd. ns-2 Network Simulator. In *Obtain via: http://www. isi. edu/nsnam/ns*.

[58] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M.B. Srivastava. Coverage Problems in Wireless Ad-hoc Sensor Networks. In *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings (INFOCOM)*, volume 3, 2001.

[59] D. Moore, J. Leonard, D. Rus, and S. Teller. Robust Distributed Network Localization with Noisy Range Measurements. In *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 50–61. ACM Press, 2004.

[60] R. Nagpal, H. Shrobe, and J. Bachrach. Organizing a Global Coordinate System from Local Information on an Ad Hoc Sensor Network. In *Proceedings of 2nd International Workshop of Information Processing in Sensor Networks (IPSN)*, Palo Alto, CA, USA, April 2003. Springer.

[61] J. Newsome, E. Shi, D. Song, and A. Perrig. The Sybil Attack in Sensor Networks: Analysis & Defenses. In *Proceedings of the third International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 259–268. ACM Press New York, NY, USA, 2004.

[62] D. Nicolescu and B. Nath. Ad-Hoc Positioning Systems (APS). In *Proceedings of the 43th IEEE Global Communications Conference(GLOBECOM)*, volume 1, pages 25–29, 2001.

[63] D. Niculescu and B. Nath. Ad Hoc Positioning System (APS) Using AOA. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 3, 2003.

[64] D. Niculescu and B. Nath. DV Based Positioning in Ad Hoc Networks. *Telecommunication Systems*, 22(1):267–280, 2003.

[65] D. Niculescu and B. Nath. Error Characteristics of Ad Hoc Positioning Systems (APS). In *Proceedings of the 5th ACM International Symposium on Mobile Ad-hoc Networking and Computing (MobiHoc)*, Tokyo, Japan, 2004.

[66] H.A.B.F. Oliveira, E.F. Nakamura, A.A.F. Loureiro, and A. Boukerche. Error analysis of localization systems for sensor networks. In *Proceedings of the 13th Annual ACM International Workshop on Geographic Information Systems (GIS)*, pages 71–78, Bremen, Germany, 2005. ACM Press.

[67] J. Ortiz, C.R. Baker, D. Moon, R. Fonseca, and I Stoica. Beacon location service: A location service for point-to-point routing in wireless sensor networks. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 166–175, Cambridge, Massachusetts, USA, 2007. ACM Press.

[68] R.K. Patro. Localization in Wireless Sensor Network with Mobile Beacons. In *Proceedings of the 23rd IEEE Convention of Electrical and Electronics Engineers*, pages 22–24, Israel, 2004.

[69] N. Patwari and A.O. Hero III. Using Proximity and Quantized RSS for Sensor Localization in Wireless Networks. In *Proceedings of the 2nd ACM international conference on Wireless Sensor Networks and Applications (WSNA)*, pages 20–29, San Diego, CA, USA, 2003. ACM Press.

[70] N. Patwari and A.O. Hero III. Indirect Radio Interferometric Localization via Pairwise Distances. In *Proceedings of 3rd IEEE Workshop on Embedded Networked Sensors (EmNets)*, pages 26–30, 2006.

[71] R. Poovendran and L. Lazos. A Graph Theoretic Framework for Preventing the Wormhole Attack in Wireless Ad Hoc Networks. *Wireless Networks*, 13(1):27–59, 2007.

[72] N.B. Priyantha, H. Balakrishnan, E. Demaine, and S. Teller. Anchor-Free Distributed Localization in Sensor Networks. In *Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 340–341, 2003.

[73] N.B. Priyantha, H. Balakrishnan, E. Demaine, and S. Teller. Mobile-Assisted Localization in Wireless Sensor Networks. In *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 1, page 172. IEEE, 2005.

[74] N.B. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket Location-Support System. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 32–43. ACM Press, 2000.

[75] A. Rao, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic Routing without Location Information. In *Proceedings of the 9th ACM International Conference on Mobile Computing and Networking (MobiCom)*, pages 96–108. ACM Press, 2003.

[76] K. Römer. The Lighthouse Location System for Smart Dust. In *Proceedings of the 1st International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 15–30. ACM Press New York, NY, USA, 2003.

[77] A.W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1998.

[78] R. Salomon. Precise Localization in Coarse-Grained Localization Algorithms Through Local Learning. In *Proceedings of the 2nd Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pages 533–540, 2005.

[79] C. Savarese, J. Rabaey, and K. Langendoen. Robust Positioning Algorithms for Distributed Ad-Hoc Wireless Sensor Networks. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pages 317–328, Berkeley, CA, USA, 2002. USENIX Association.

[80] A. Savvides, C.C. Han, and M.B. Strivastava. Dynamic Fine-Grained Localization in Ad-Hoc Networks of Sensors. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 166–179. ACM Press, 2001.

[81] Y. Shang, W. Rumi, Y. Zhang, and M. Fromherz. Localization from Connectivity in Sensor Networks. *IEEE Transactions on Parallel and Distributed Systems*, 15(11):961–974, 2004.

[82] Y. Shang and W. Ruml. Improved MDS-based Localization. In *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 4, 2004.

[83] Y. Shang, W. Ruml, Y. Zhang, and M.P.J. Fromherz. Localization from Mere Connectivity. In *Proceedings of the 4th ACM International Symposium on Mobile Ad-hoc Networking and Computing (MobiHoc)*, pages 201–212. ACM Press, 2003.

[84] B. Sheng, Q. Li, W. Mao, and W. Jin. Outlier Detection in Sensor Networks. In *Proceedings of the 8th ACM International Symposium on Mobile Ad-hoc Networking and Computing (MobiHoc)*, pages 219–228. ACM Pres, 2007.

[85] S.E. Smaha. Haystack: An Intrusion Detection System. In *Proceedings of the Fourth Aerospace Computer Security Applications Conference*, pages 37–44, Austin, TX, USA, 1988.

[86] A. Smith, H. Balakrishnan, M. Goraczko, and N. Priyantha. Tracking moving devices with the cricket location system. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 190–202. ACM Press New York, NY, USA, 2004.

[87] A.M.C. So and Y. Ye. Theory of semidefinite programming for Sensor Network Localization. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 405–414. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 2005.

[88] R. Stoleru, T. He, J.A. Stankovic, and D. Luebke. Spotlight: A High Accuracy, Low-Cost Localization System for Wireless Sensor Networks. In *Proceedings of the 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 13–26. ACM Press New York, NY, USA, 2005.

[89] H.S. Teng, K. Chen, and S.C. Lu. Adaptive Real-time Anomaly Detection Using Inductively Generated Sequential Patterns. In *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 278–284, 1990.

[90] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A Macroscope in the Redwoods. In *Proceedings of the 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 51–63. ACM Press, 2005.

[91] C.F. Tsai. Pyroelectric Infrared Sensor-based Thermometer for Monitoring Indoor Objects. *Review of Scientific Instruments*, 74(12):5267, 2003.

[92] P. Vicaire and J.A. Stankovic. Elastic Localization: Improvements on Distributed, Range Free Localization for Wireless Sensor Networks. Technical Report SRI Annual Report A010, Tech. Rep. CS-2004-35, University of Virginia, 2004.

[93] M.A.M. Vieira, C.N. Coelho Jr, D.C. da Silva Jr, and J.M. da Mata. Survey on Wireless Sensor Network Devices. In *Proceedings of IEEE Emerging Technologies and Factory Automation (ETFA)*, pages 537–544, 2003.

[94] G. Virone, A. Wood, L. Selavo, Q. Cao, L. Fang, T. Doan, Z. He, R. Stoleru, S. Lin, and J.A. Stankovic. An Assisted Living Oriented Information System Based on a Residential Wireless Sensor Network. In *Proceedings of 1st Transdisciplinary Conference on Distributed Diagnosis and Home Healthcare (D2H2)*, pages 95–100. IEEE Press, 2006.

[95] D. Wagner and R. Dean. Intrusion Detection via Static Analysis. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 156–168, 2001.

[96] C.Y. Wan, A.T. Campbell, and L. Krishnamurthy. PSFQ: A Reliable Transport Protocol for Wireless Sensor Networks. *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 1–11, 2002.

[97] W. Wang and B. Bhargava. Visualization of Wormholes in Sensor Networks. In *Proceedings of ACM workshop on Wireless Security (WiSe)*, pages 51–60. ACM Press New York, NY, USA, 2004.

[98] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: alternative data models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 133–145, 1999.

[99] A.D. Wood and J.A. Stankovic. Denial of Service in Sensor Networks. *Computer*, 35(10):54–62, 2002.

[100] Y. Xu, E. Becker, J. Ford, and F.S. Makedon. BP-Neural Network Improvement to Hop-counting for Localization in Wireless Sensor Networks. In *Proceedings of International Workshop on Applications with Artificial Intelligence (AAI), conjuncted in conjunction with The 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, Patras, Hellas, Greece, October 2007. Springer.

[101] Y. Xu, G. Chen, J. Ford, and F.S. Makedon. Distributed Wormhole Detection in Wireless Sensor Networks. In Eric Goetz and Sujeet Shenoi, editors, *Book Series of Critical Infrastructure Protection: Issues and Solutions*, chapter 19, pages 267–279. Springer, 2007.

[102] Y. Xu, J. Ford, and F.S. Makedon. A Variation on Hop-counting for Geographic Routing. In *Proceedings of the 3rd IEEE Workshop on Embedded Networked Sensors (EmNetS)*, April 2006.

[103] Y. Xu, J. Ford, and F.S. Makedon. GDL: A Geographic Distributed Localization Algorithm for Wireless Sensor Networks. In *Proceedings of the 15th International Conference on Computer Communications and Networks (ICCCN)*, Arlington, VA, USA, October 2006.

[104] Y. Xu, J. Ford, and F.S. Makedon. Mobile Anchor-free Localization for Wireless Sensor Networks. In *Proceedings of the 3rd IEEE International Distributed Computing in Sensor Systems (DCOSS), LNCS 4549*, pages 96–109, Santa Fe, NM, USA, June 2007. Springer.

[105] Y. Xu, Y. Ouyang, Z. Le, J. Ford, and F.S. Makedon. Analysis of Range-Free Anchor-Free Localization in a WSN under Wormhole Attack. In *Proceedings of the 10th ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, Chania, Crete Island, Greece, October 2007. ACM Press.

[106] J. Zheng and Lee J. M. 802.15.4 Extension to NS-2. In *Obtain via http://www-ee.ccny.cuny.edu/zheng/pub*.

[107] G. Zhou, T. He, S. Krishnamurthy, and J.A. Stankovic. Impact of Radio Irregularity on Wireless Sensor Networks. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 125–138. ACM Press, 2004.